1990

Storage capacities and applications of some neural networks.

Chi Kin. Lee
University of Windsor

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

STORAGE CAPACITIES AND APPLICATIONS OF SOME NEURAL NETWORKS

by

Chi Kin Lee

A Thesis
Submitted to the
Faculty of Graduate Studies and Research
through the Department of
Electrical Engineering in Partial Fulfillment
of the Requirements for the Degree
of Master of Applied Science at the
University of Windsor

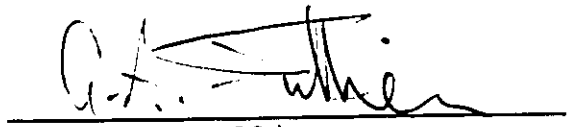
Windsor, Ontario, Canada

1990

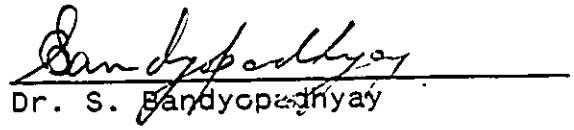
ISBN 0-315-60984-2

APPROVED BY:

Dr. H. K. Kwan
Supervisor



Dr. G. A. Jullien



Dr. S. Bandyopadhyay

To my family

ACKNOWLEDGMENTS

I would like to express my sincere thanks and appreciation to Dr. H. K. Kwan for his generous advice and guidance throughout the progress of my research. In addition, I would also like to thank Dr. G. A. Jullien and Dr. S. Bandyopadhyay for their comments on my seminars and thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I INTRODUCTION

1.1 WHAT ARE NEURAL NETWORKS?

Artificial neural networks, also known as parallel distributed processing models, attempt to achieve good performance in computation via dense interconnections of simple processing units. Instead of performing a program of instructions sequentially as in a von Neumann computer, neural networks perform computation simultaneously using massively parallel networks composed of many processing units connected by links with variable weights. In this respect, a neural network is based on our present understanding of biological nervous systems, i.e., a collection of nerve cells, or neurons (Fig. 1.1), each of which is connected to others, from which it receives stimuli--inputs and feedback -- and to which it sends stimuli. In Fig. 1.1, the nucleus is a simple processing unit which receives and combines signals from many other neurons through input paths called dendrites. The nucleus sends the output signal, which has a value depending on the input signals, to other neurons through output paths called axons. The junction between the axon and the dendrite is called synapse. The synaptic

strength, i.e., the amount of signal transferred, is modifiable through learning and is the basic memory unit of the brain.

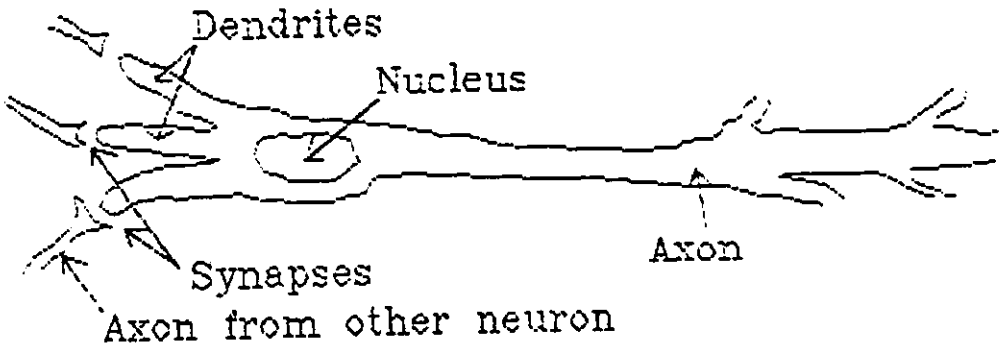


Fig. 1.1 A neuron in the nervous system

Artificial neural networks have a similar structure and have the following eight major aspects in modeling[17]:

1. The processing units:

Typically, specifying the set of processing units, or neurons, and what they represent is the first stage of specifying a neural network model. They may represent particular elements in a pattern or may represent a class of object. Each neuron is a very simple unit, it simply receives input from its neighbors and, as a function of the inputs it receives, it computes an output value which it

sends to its neighbors. The network is parallel in that many processing units can carry out their computations at the same time. Within each network, there are three types of processing units: input, hidden and output. Input units receive signals from external sources, hidden units have their inputs and outputs within the network (thus invisible to the external system) while output units send their signals to the external system.

2. The state of activation:

Different networks make different assumptions about the activation values that a unit can take. They can be continuous with or without boundary or they can be discrete in binary, bipolar or a finite set of integer values.

3. Output of the units:

Units interact by sending their output signals to other units. These outputs are functions of their activations, and usually, the function is a threshold function.

4. Pattern of connectivity:

This pattern of connectivity determines what the system knows and how it will respond to any arbitrary input. In many cases, the total input of a unit is simply the weighted sum of the outputs of other units that connect to it. Thus, the pattern of connectivity can be represented by specifying

the weights for each of the connections in the system. A positive weight represents an excitatory input and a negative weight represents an inhibitory input.

5. Propagation rule:

This is the rule which takes the outputs of the units and combines them with the weight matrix to produce a net input for each unit. In a feed-forward network, the units in one layer only take the outputs from the units in the lower layer.

6. Activation rule:

This rule takes the total inputs and current activation state to produce a new state of activation. Usually, the function is assumed to be deterministic and sometimes the activations are assumed to decay slowly with time so that even with no external input, the activation of a unit will simply decay and not go directly to zero. A processing unit and some common types of activation functions are shown in Fig. 1.2.

7. Learning rule:

The network learns to solve a problem by the modifications of the pattern of connectivity. There are three kinds of modifications:

(a) The development of new connections.

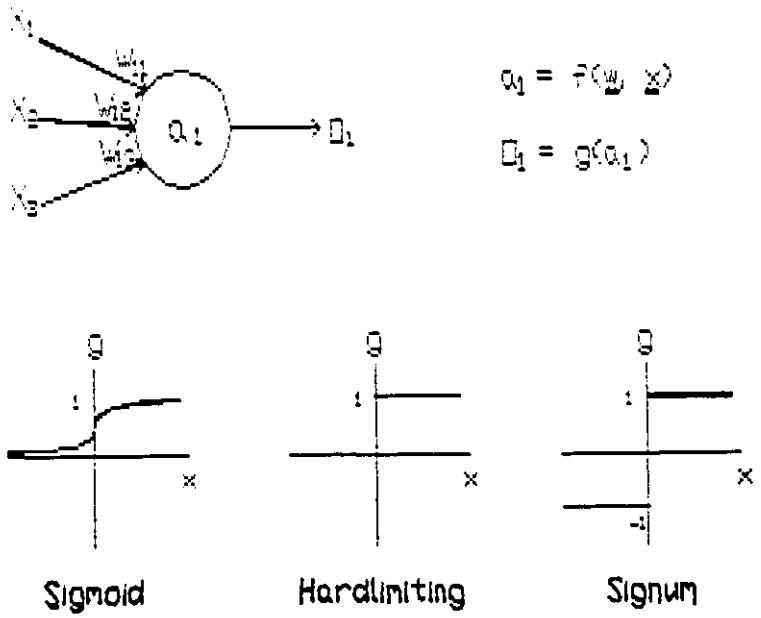


Fig. 1.2 A typical processing unit and some activation functions.

(b) The loss of existing connections.

(c) The modification of the strengths of connections that already exist.

(a) and (b) can be considered a special case of (c) and so only (c) will be discussed. The basic rule is suggested by Hebb in his book "Organization of Behavior" (1949)[4]. The basic idea is that if a unit, u_i, which is highly active, receives an input from another highly active unit, u_j, the weight, w_{ij}, from u_j to u_i should be strengthened. This idea has been extended and modified so that it can be more generally stated as in equation (1.1).

$$\delta w_{ij} = g(a_i(t), t_i(t))h(o_j(t), w_{ij}) \qquad (1.1)$$

where $t_i(t)$ is a kind of teaching input to u_i, $a_i(t)$ is the activation of u_i, o_j is the output of u_j and g(.) and h(.) are some functions. In the simplest versions of Hebbian learning, there is no teacher and the functions g and h are simply proportional to their first arguments (please refer to Chapter 2 for the equation). Another common rule is the Widrow-Hoff rule[19] which is often known as the delta rule because the amount of learning is proportional to the difference (or delta) between the actual activation achieved and the target activation provided by a teacher as in equation (1.2).

$$\delta w_{ij} = lrate*(t_i(t)-a_i(t))*o_j(t) \qquad (1.2)$$

where lrate is the learning rate parameter controlling the rate of learning.

8. Representation of the environment:

The environment can be considered a time-varying stochastic function over the space of input patterns, i.e., at any point in time, there is some probability that any of the possible set of input patterns is presented to the input units. Typically, the environment is characterized by a stable probability distribution over the set of possible input patterns independent of past inputs and the past response of the system. In this case, the possible set of input patterns can be listed associated with a set of probabilities.

1.2 WHY NEURAL NETWORKS?[10]

Work on artificial neural networks has a long history. Development of detailed mathematical models began more than 40 years ago with the work of McCulloch and Pitts[11], Hebb[4], Rosenblatt[16] and Widrow[19]. However, study by Minsky and Papert in 1969[14] showed that the single layer perceptron by Rosenblatt, which is a feed-forward network without any feedback and whose neurons used a linear threshold activation function, cannot solve some simple mathematical functions such as XOR. Moreover, they pointed out that though a multi-layer perceptron might be able to solve the problem, there was no mechanism for figuring out which weights to adjust when the output was incorrect. This reduced the effort into neural network research. Recently, important research work is done by Hopfield[5, 6, 7], Rumelhart and McClelland[17], Grossberg[3] and Kohonen[8]. This new interest is due to the development of new network topologies and algorithms[5, 6, 17] and VLSI implementation techniques, and some intriguing demonstrations[7] as well as by a growing fascination with the functioning of the human brain. Recent interest is also driven by the realization that human-like performance in the areas of speech and image recognition will require enormous amounts of processing. Neural networks provide one technique for obtaining the required processing capacity using large numbers of simple

processing units operating in parallel.

Besides providing the high computation rates by massive parallelism, neural networks also provide a greater degree of robustness or fault tolerance than von Neumann sequential computers because there are many more processing units, each with primarily local connections. Damage to a few units or links thus need not impair overall performance significantly. In addition to this, most neural network algorithms adapt connection weights to improve performance based on current results. This ability to adapt and continue learning is essential in areas such as speech recognition where training data is limited and new talkers, new words, new dialects and new environments are continuously encountered. Neural network classifiers are also non-parametric and make weaker assumptions concerning the shapes of underlying distributions than traditional statistical classifiers[10]. They may thus prove to be more robust when distributions are generated by nonlinear processes and are strongly non-Gaussian.

1.3 A TAXONOMY OF NEURAL NETWORKS[10]

A taxonomy of some important neural networks is presented in Fig. 1.3. This taxonomy is first divided between networks with binary and continuous valued inputs. Below this, networks are divided between those trained with and without supervision. Networks trained with supervision such as the Hopfield network[5, 6], the Bidirectional Associative Memories[9] and the perceptrons[16] are used as associative memories or as classifiers. These networks are provided with information or labels that specify the correct class for new input patterns during training. Most traditional statistical classifiers, such as Gaussian classifiers[2], are trained with supervision using labeled training data. For the networks that are trained without supervision, such as the Kohonen's feature-map forming network[8], no information concerning the correct class is provided during training. In this thesis, only the supervised networks are considered.

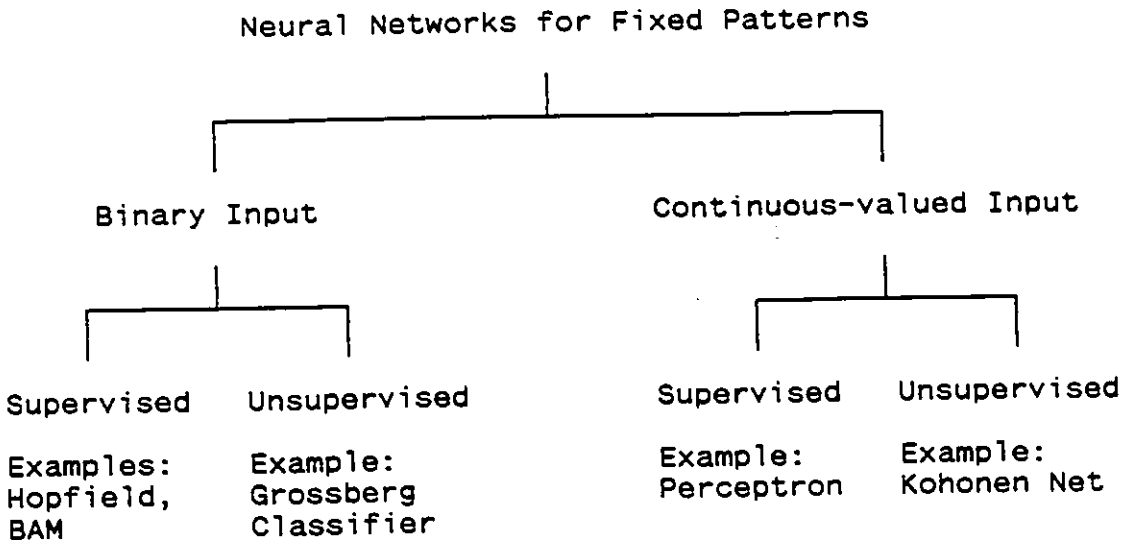


Fig. 1.3 A taxonomy of some important neural networks.

1.4 THESIS ORGANIZATION

In this thesis, Chapter I, INTRODUCTION, discusses the basic knowledge in neural network. Some important aspects and the advantages of using neural network are discussed. Chapter II discusses the details of three neural networks, i.e., the Hopfield network, the Bidirectional Associative Memories and the multi-layer feed-forward network, which are to be studied in this thesis. Chapter III is the first part of my specific research work and investigates about the storage capacities of neural networks. The limitations of current neural networks are discussed together with new methods for improving pattern storage. A comparison is made among different methods. The second part is to be discussed in Chapter IV and is about the applications of the multi-layer feed-forward network. Two special applications are discussed, one is in pulse compression and the other is in filtering. Simulations were done and the results are presented. In Chapter V, CONCLUSIONS, my research works are summarized and recommendations for further study are given.

CHAPTER II CLASSICAL NETWORKS

2.1 THE HOPFIELD NETWORK

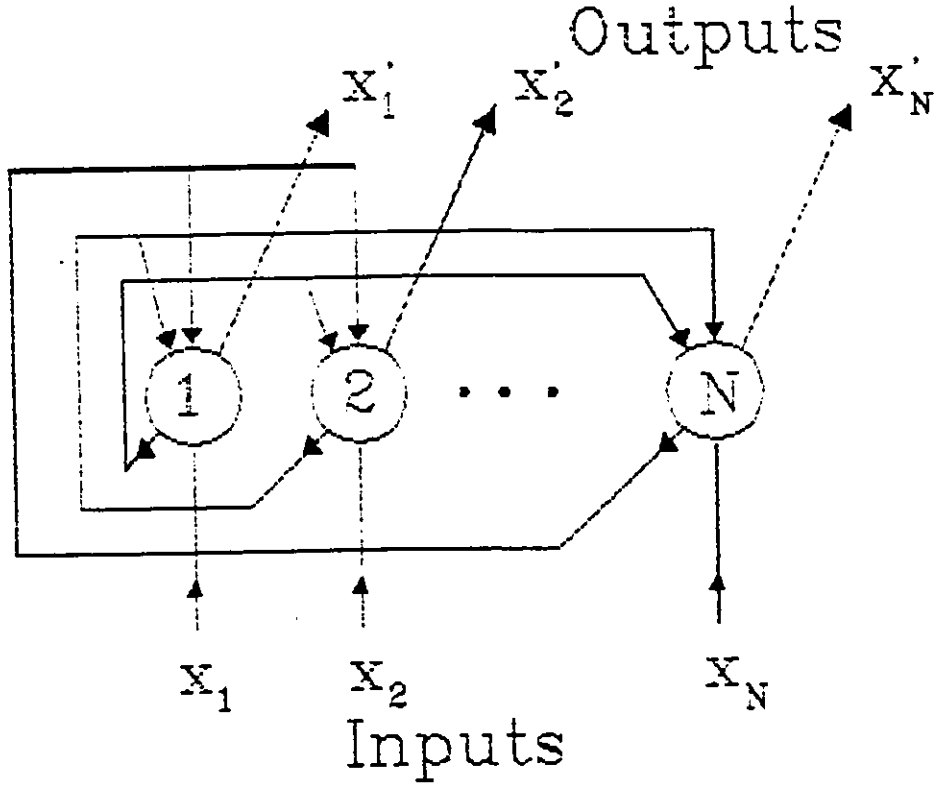


Fig. 2.1 The Hopfield network.

The Hopfield network is usually used with binary inputs. It is very suitable when exact binary representations are possible such as with black and white images where input elements are pixel values, or with ASCII text where input values could represent bits in the ASCII representation of

each character. There are different versions of the Hopfield network[5, 6]. Here, only one of them will be presented. This version can be used as a content addressable memory and is shown in Fig. 2.1. It consists of N neurons which contain the hardlimiting function. Bipolar inputs and outputs are used and the output of each neuron is fed back to all other neurons via weights denoted w_{ij}.

The training of this network is done by the Hebb rule and the self-feedback weight is zero, i.e.,

$$w_{ij} = \begin{cases} \sum_{s=1}^{M} x_i^s x_j^s & 1 \le i, j \le N \text{ and } i \ne j \\ 0 & 1 \le i, j \le N \text{ and } i = j \end{cases} \qquad (2.1)$$

where w_{ij} is the value of the weights connecting neuron j to neuron i; x_i^s is the ith element of the sth pattern. Altogether, there are M patterns and each pattern has N elements.

The recalling of this network is that an unknown pattern is presented to the network at time zero by forcing the output of all the neurons the same as the pattern. Then the following formula is used to update the output of the neurons until there is no change in the output states on successive iterations:

$$O_i(t+1) = H\left[\sum_{j=1}^{N} w_{ij} O_j(t)\right] \qquad 1 \le i \le N \qquad (2.2)$$

where $O_i(t)$ is the output state of neuron i at time t; H[.] is the hardlimiting function as shown in equation (2.3):

$$H[z_i] = \begin{cases} 1 & \text{if } z_i \geq U_i \\ 0 & \text{if } z_i < U_i \end{cases} \qquad (2.3)$$

Hopfield formulated the dynamics of the network in terms of Spin Glass physics[5, 6]. This is the result of viewing the state of the network as an energy surface. The energy of the system is:

$$E = -1/2 \sum_{i \neq j} \sum w_{ij} x_i x_j + \sum_i U_i x_i \qquad (2.4)$$

It can be guaranteed that the state space flow algorithm converges on stable states if the weight is symmetric with zero diagonal elements and the states change one at a time in some random order. The proof[5, 6] is as follows:
The change of energy due to the changing of the state of neuron i is given by:

$$\delta E_i = -\left(\sum_{i \neq j} w_{ij} x_j - U_i \right) * \delta x_i \qquad (2.5)$$

If x_i is currently 0 and it becomes 1, δx_i is positive, then the bracket in equation (2.5) is positive according to equation (2.3). Thus δE_i is negative. If it stays at 0, δx_i is zero and thus there is no change in energy. Likewise, if x_i is currently 1 and becomes 0, and hence δx_i is negative, the bracket in equation (2.5) is again negative according to equation (2.3). Thus again, δE_i is negative. If x_i stays at 1, δx_i is zero and thus there is no change in energy. As

a result, an element in the network will only change its state if, and only if, it will decrease the overall energy of the network. Since E is bounded by the finite summation of each product term, the network converges to stable states that do not further change with time.

2.2 THE BIDIRECTIONAL ASSOCIATIVE MEMORIES

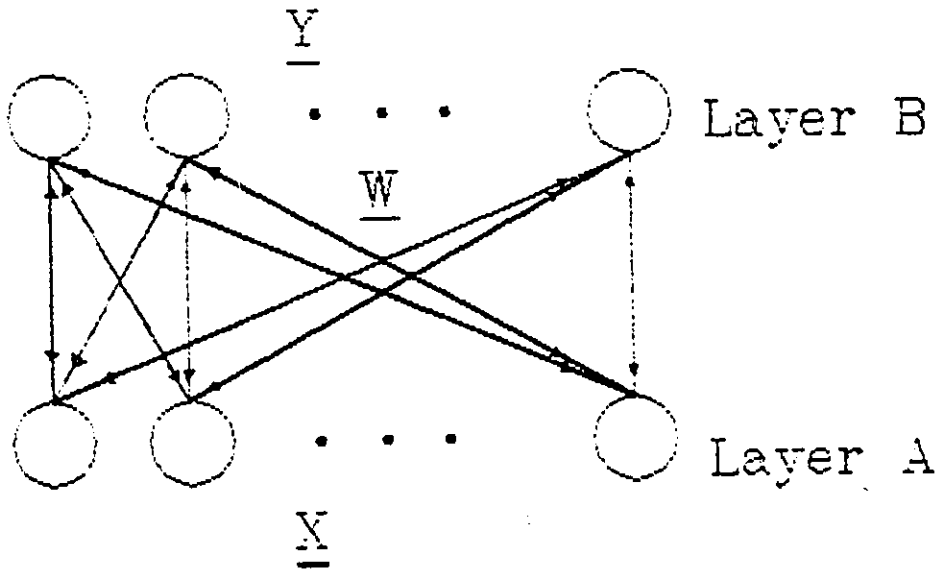


Fig. 2.2 The BAM.

Bidirectional Associative Memories (BAM)[9] are very similar to Hopfield networks except that the Hopfield network is used for auto-associative data only, i.e., the trained inputs and the target outputs are the same while the BAM can be used for both the auto-associative and the hetero-associative data, i.e. the trained inputs and the target outputs can be different. As shown in Fig. 2.2, a Hopfield network can be constructed from a BAM by making the upper neurons equal to the lower neurons and setting the self-connection weights equal to zero.

In this network, the pattern pair does not need to have the same length. The training of this network is also done by the Hebb rule and in vector notation, it is given by equation (2.6).

$$\underline{W} = \sum_{s=1}^{M} \underline{X}^T{}_s \underline{Y}_s \qquad (2.6)$$

where, if N is the length of patterns $\underline{X}$ and P is the length of patterns $\underline{Y}$, $\underline{W}$ will be a matrix of PxN. Here, M is the total number of patterns stored.

Data stored can be recalled by the following equations:

$$\underline{Y} = H[\ \underline{X}\ \underline{W}\ ] \qquad (2.7)$$

$$\underline{X} = H[\ \underline{Y}(\underline{W})^T] \qquad (2.8)$$

where H[] is the hardlimiting function. The operations of this BAM are: first apply $\underline{X}$' in to layer A in Fig. 2.2 and then recall $\underline{Y}$' in layer B using equation (2.7) in one cycle. Then $\underline{Y}$' will recall back $\underline{X}$'' using equation (2.8) in the next cycle and so on until both outputs are stable to a final pair ($\underline{X}^f$, $\underline{Y}^f$). The proof of stability of this network is similar to the Hopfield network and thus is omitted here.

This network is used as a content addressable associative memory and can be used for data compression when one data length is shorter than the other.

2.3 THE BACK-PROPAGATION NETWORK

The multi-layer feed-forward network using back-propagation training[17] is one of the most powerful neural networks for classification. The structure is as shown in Fig. 2.3.

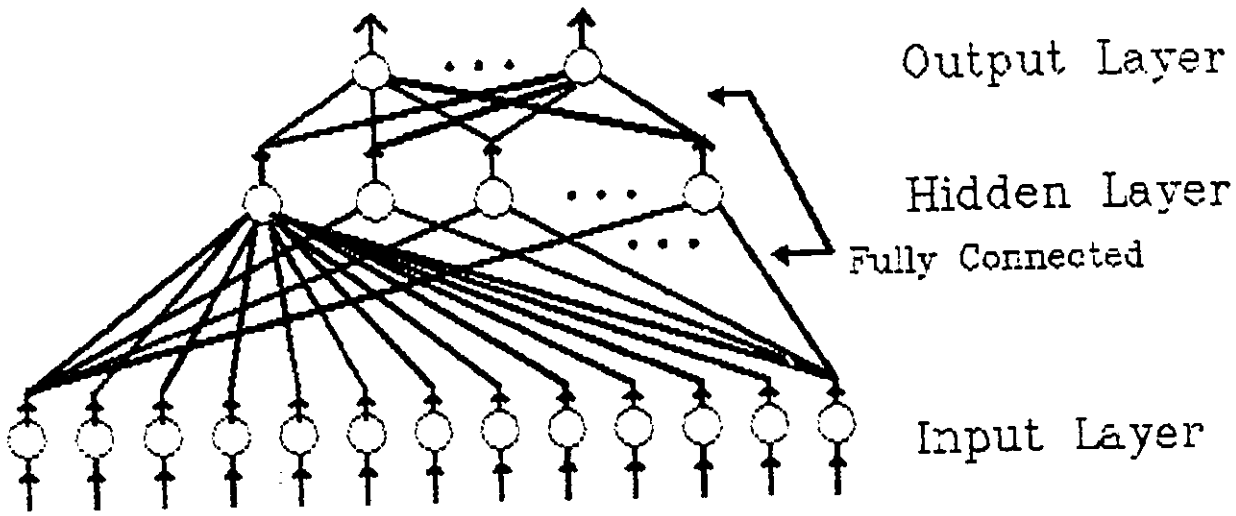


Fig. 2.3 The multi-layer feed-forward network.

This network usually consists of one input layer, one to two hidden layer(s) and one output layer. Full or random set of weights are used for connecting two layers.

The back-propagation rule is as follows: Suppose we have an L-layer network, starting from layer 0 to layer L-1. There are N_k neurons in the kth layer. The pth input vec-

tor of length N is presented to the network to train the network. During the training, the outputs of all the neurons are first calculated, starting at the input layer, then the hidden layer(s), and finally the output layer, using equations (2.9) to (2.11).

For the input layer, k = 0,

$$o_{pi} = i_{pi} \tag{2.9}$$

where i_{pi} is the ith element of the pth input vector; and o_{pi} is the output of the ith neuron in the kth layer.

For the other layer, k = 1 to L-1,

$$net_{pj}[k] = \sum_{i=1}^{N_{k-1}} w_{ji}[k]\, o_{pi}[k-1] + t_j[k] \tag{2.10}$$

and

$$o_{pi}[k] = f(\, net_{pi}[k] \,) \tag{2.11}$$

where $w_{ji}[k]$ is the weight connecting the ith neuron in the (k-1)th layer to the jth neuron in the kth layer and should be initialized to zero or a small random value; $t_j[k]$ is the threshold value of the jth neuron in the kth layer; $net_{pj}[k]$ is the activation of the jth neuron in the kth layer; and f(.) is the activation function which must be a differentiable function. Some common activation functions are:

a) Sigmoid function:

The sigmoid function is defined as

$$f(z) = 1/(1+e^{-z}) \tag{2.12}$$

and the derivative of this function can be expressed as

$$f'(z) = f(z) * (1 - f(z)) \qquad (2.13)$$

b) Hyperbolic tangent:

The hyperbolic tangent is defined as

$$f(z) = (1 - e^{-z}) / (1 + e^{-z}) \qquad (2.14)$$

and its derivative can be expressed as

$$f'(z) = 0.5 * (1 + f(z)) * (1 - f(z)) \qquad (2.15)$$

c) Sine function:

When a sine function is used, the learning procedure seems to perform a mode decomposition in that it discovers the important components of the function described by the discrete set of input/output examples[20].

d) Linear function:

In this case, it can be shown that there is no additional advantage to be gained in using hidden layers and the learning procedure reduces to the Widrow-Hoff or the delta rule for a Linear Associator[17]. The delta rule for binary and bipolar data in an associator will be discussed in Chapter III.

After the forward propagation of the input through the layers to the output, the error between the actual output and the desired output is determined and the errors are then propagated back through the network from the output layer to

the input layer. The errors at each neuron are calculated using equations (2.16) or (2.17).

For the output layer, k = L-1,

$$e_{pj}[L-1] = f'(net_{pj}[L-1]) * (d_{pj} - o_{pj}[L-1]) \quad (2.16)$$

For the other layer, k = L-2 to 1,

$$e_{pj}[k] = f'(net_{pj}[k]) * \sum_{i=1}^{N_{k+1}} e_{pi}[k+1] * w_{ij}[k+1] \quad (2.17)$$

where $e_{pj}[k]$ is the error at the jth neuron in the kth layer; f'(.) is the derivative of the activation function; and d_{pj} is the jth element of the desired output.

After all errors have been calculated, the weights are updated by adding a delta value given by equation (2.18):

$$\delta w_{ij}[k] = lrate * e_{pi}[k] * o_{pj}[k-1] \quad (2.18)$$

where lrate is the learning rate parameter which controls the rate of learning convergence. Another input pattern will then be presented to the network for training. After the whole set of patterns are trained, this is called one training epoch, the whole process will be repeated until some criteria, such as the maximum root mean square error of the outputs is smaller than a certain value, are met.

The above is a gradient descent algorithm, which searches the minimum value by following the direction which can reduce the energy function, it makes the assumption that

the error surface is locally linear, where "locally" is defined by the size of the learning rate parameter. At points of high curvature, this assumption does not hold and divergent behavior might occur. So it is important to keep the learning rate parameter small.

On the other hand, a small learning rate parameter will lead to very slow learning. So a momentum term[20] can be introduced to solve this dilemma as in equation (2.19).

$$\delta w_{ij}[k]_{new} = lrate * e_{pi}[k] * o_{pj}[k-1] + momentum * \delta w_{ij}[k]_{old} \quad (2.19)$$

Another technique is to update the weights after a whole set of (input, desired output) pairs is presented to the network, rather than after every pattern presentation. This is called a cumulative learning[20].

Back-propagation is a very powerful training rule and can have analogue inputs and outputs. So it has many interesting applications. Some of them are data encoding and compression, signal prediction and pattern classification. In this thesis, two new applications will be discussed in Chapter IV.

CHAPTER III STORAGE CAPACITIES

3.1 LIMITATIONS OF CURRENT NETWORKS

For storing associative data, the Hopfield network (for auto-associative data) and the BAM (for hetero-associative data) are always used since they are simple. Both networks use the Hebb rule for training the weights. The use of Hebb rule has a great limitation in that the storage capacity is very limited. The storage capacity is defined as the number of stable states, selected by the user at random, that can be implemented by a network for some weight matrix selection and some set of thresholds. Here, stable states are the states that will never change once they are achieved.

Experimentally speaking, the storage capacity of the Hopfield network is 0.15N where N is the number of neurons[5]. Statistically, it can be shown that if even-coded vectors are stored, the storage capacity is $N/(2 \log_2 N)$[12]. These values also apply to the BAM where N is now the minimum of the two vector lengths.

In this chapter, some methods for improving the storage capacity of these networks will be discussed.

Another aspect in storing associative data is the problem of storing temporal data, the Temporal Associative Memories (TAM). A good example is a melody, which consists of a set of ordered vectors. If a single BAM is used for storing temporal data, for example, ($\underline{X}_1$, $\underline{X}_2$, $\underline{X}_3$, $\underline{X}_4$), we can store the data pairs: ($\underline{X}_1$, $\underline{X}_2$), ($\underline{X}_2$, $\underline{X}_3$), ($\underline{X}_3$, $\underline{X}_4$) and ($\underline{X}_4$, $\underline{X}_1$)[9]. Then the whole sequence can be recalled by inputting $\underline{X}_1$ and then feeding back the output $\underline{X}_2$ to the input to recall $\underline{X}_3$ in the next iteration and so on. In this method, the length of this sequence will be limited by the storage capacity of the BAM and also we cannot store a sequence which has identical vectors in different positions, for example, we cannot store ($\underline{X}_1$, $\underline{X}_2$, $\underline{X}_1$, $\underline{X}_3$) or ($\underline{X}_1$, $\underline{X}_1$, $\underline{X}_2$, $\underline{X}_3$). The solution of this problem will be discussed in the next section.

3.2 USE OF RING AND CASCADE NETWORKS

3.2.1 TAM Using a Single BAM

In order to store temporal associative patterns, we can use two different weight matrices for the two opposite directions in a BAM. For example, if we want to store ($\underline{X}_1$, $\underline{X}_2$, $\underline{X}_3$, $\underline{X}_4$), we can store ($\underline{X}_1$, $\underline{X}_2$) and ($\underline{X}_3$, $\underline{X}_4$) in one direction (say forward direction), and ($\underline{X}_2$, $\underline{X}_3$) and ($\underline{X}_4$, $\underline{X}_1$) in the other direction (say backward direction). Then we can recall $\underline{X}_2$ by inputting $\underline{X}_1$ in the forward direction and $\underline{X}_2$ will then recall $\underline{X}_3$ in the backward direction and so on as shown in Fig. 3.1.

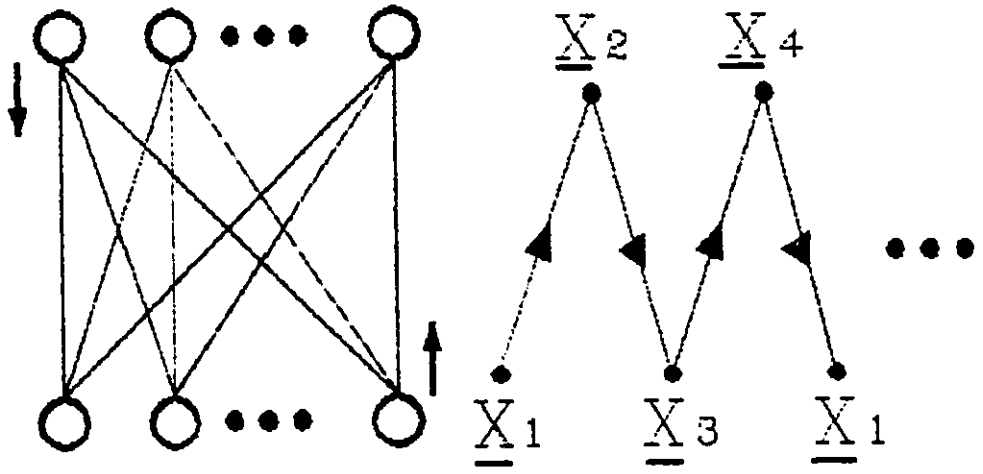


Fig 3.1 Temporal associative memories using a single BAM.

In this method, the bidirectional recalling is utilized

and no feedback path is needed to do the next iteration once there is an output in the BAM as described before. In general, the weight matrices (i.e., $\underline{W}_f$ for the forward direction and $\underline{W}_b$ for the backward direction) for the storage of L vectors are calculated as follows:

$$\underline{W}_f = (\underline{X}_1)^T\underline{X}_2 + (\underline{X}_3)^T\underline{X}_4 + \ldots + (\underline{X}_{L-1})^T\underline{X}_L \quad (3.1)$$

$$\underline{W}_b = (\underline{X}_2)^T\underline{X}_3 + (\underline{X}_4)^T\underline{X}_5 + \ldots + (\underline{X}_L)^T\underline{X}_1 \quad (3.2)$$

$$\underline{X}_{2i} = H[\ \underline{X}_{2i-1}\underline{W}_f\ ] \text{ where } i = 1, 2, \ldots, L/2 \quad (3.3)$$

$$\underline{X}_{2i+1} = H[\ \underline{X}_{2i}\underline{W}_b\ ] \quad \text{where } i = 1, 2, \ldots, L/2-1 \quad (3.4)$$

$$\underline{X}_1 = H[\ \underline{X}_L\underline{W}_b\ ] \quad (3.5)$$

where H[.] is the hardlimiting function.

In order to store N (≥2) such sequences of patterns, $\underline{X}_{1k}$, $\underline{X}_{2k}$, ... , $\underline{X}_{jk}$, ... , $\underline{X}_{Nk}$ where $\underline{X}_{jk}$ is one set of temporal pattern with elements $(\underline{X}_{j,1}, \underline{X}_{j,2}, \ldots, \underline{X}_{j,L})$, we can modify the weight matrices by the following equations:

$$\underline{W}_f = (\underline{X}_{1,1})^T\underline{X}_{1,2} + \ldots + (\underline{X}_{1,L-1})^T\underline{X}_{1,L} + (\underline{X}_{2,1})^T\underline{X}_{2,2} + \ldots + (\underline{X}_{2,L-1})^T\underline{X}_{2,L} + \ldots + (\underline{X}_{N,1})^T\underline{X}_{N,2} + \ldots + (\underline{X}_{N,L-1})^T\underline{X}_{N,L} \quad (3.6)$$

$$\underline{W}_b = (\underline{X}_{1,2})^T\underline{X}_{1,3} + \ldots + (\underline{X}_{1,L})^T\underline{X}_{1,1} + (\underline{X}_{2,2})^T\underline{X}_{2,3} + \ldots + (\underline{X}_{2,L})^T\underline{X}_{2,1} + \ldots + (\underline{X}_{N,2})^T\underline{X}_{N,3} + \ldots + (\underline{X}_{N,L})^T\underline{X}_{N,1} \quad (3.7)$$

Hence, we can store many temporal patterns using this method. Since we distinguish the two directions in recalling, the same data can be used when one is in the odd posi-

tion and the other is in the even position. However, in this method, the length of the temporal pattern is still limited by the storage capacity of the BAM and hence it cannot store many patterns. In order to overcome this problem, the CBAM and the RUAM are introduced.

3.2.2 Cascade Bidirectional Associative Memories (CBAM)

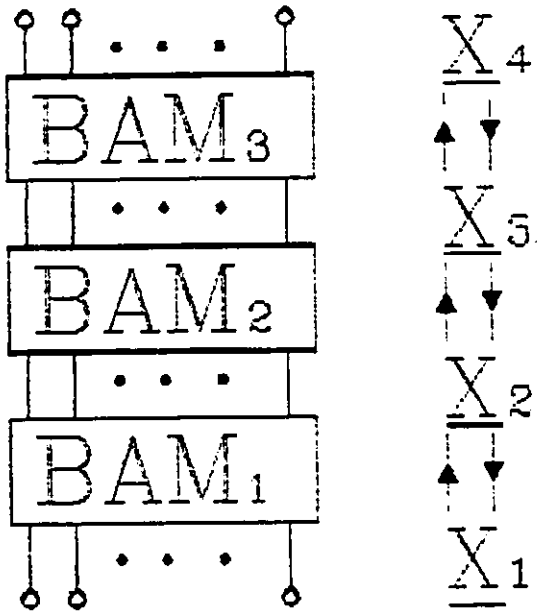


Fig. 3.2 Temporal associative memories using the CBAM.

A CBAM is constructed by connecting several BAMs to form a TAM as shown in Fig. 3.2. Given ($\underline{X}_{i,1}$, $\underline{X}_{i,2}$, ... , $\underline{X}_{i,L}$) where i = 1, 2, ... , N, we can use L-1 BAMs to store ($\underline{X}_{i,1}$, $\underline{X}_{i,2}$), ($\underline{X}_{i,2}$, $\underline{X}_{i,3}$), ... , ($\underline{X}_{i,L-1}$, $\underline{X}_{i,L}$). The

weight matrices are calculated as follows.

$$\begin{aligned}
\underline{W}_1 &= (\underline{X}_{1,1})^T \underline{X}_{1,2} + (\underline{X}_{2,1})^T \underline{X}_{2,2} + \ldots + (\underline{X}_{N,1})^T \underline{X}_{N,2} \\
\underline{W}_2 &= (\underline{X}_{1,2})^T \underline{X}_{1,3} + (\underline{X}_{2,2})^T \underline{X}_{2,3} + \ldots + (\underline{X}_{N,2})^T \underline{X}_{N,3} \\
&\vdots \\
\underline{W}_{L-1} &= (\underline{X}_{1,L-1})^T \underline{X}_{1,L} + (\underline{X}_{2,L-1})^T \underline{X}_{2,L} + \ldots + (\underline{X}_{N,L-1})^T \underline{X}_{N,L}
\end{aligned} \tag{3.8}$$

The recalling procedure is that firstly, vector $\underline{X}_{i,1}$ is input into BAM_1 and $\underline{X}_{i,2}$ $(=H[\underline{X}_{i,1}\underline{W}_1])$ will be recalled. $\underline{X}_{i,2}$ will recall $\underline{X}_{i,3}$ $(=H[\underline{X}_{i,2}\underline{W}_2])$ through BAM_2 at the next iteration and so on until $X_{i,L}$ $(=H[\underline{X}_{i,L-1}\underline{W}_{L-1}])$ is recalled. Then this vector will be passed back to BAM_{L-1} so that $\underline{X}_{i,L-1}$ $(=H[\underline{X}_{i,L}(\underline{W}_{L-1})^T])$ will be renewed. This renewal will be carried on until $\underline{X}_{i,1}$ $(=H[\underline{X}_{i,2}(\underline{W}_1)^T])$ is renewed. Then the above procedure is repeated until all neuron states are stable.

3.2.3 Ring Unidirectional Associative Memories (RUAM)

A RUAM is constructed by connecting several unidirectional associative memories (UAMs) in a ring and it can do the same job as the CBAM. A UAM is a BAM in which the signal flow is restricted to a fixed direction and is not allowed in the reversed direction. Here, the number of UAMs used is L instead of L-1 for the case of the CBAM. The structure of a RUAM is shown in Fig. 3.3. The weight matrices are the same as in equation 12 except that there is

one more weight matrix for the storing of $(\underline{X}_{i,L}, \underline{X}_{i,1})$, hence,

$$\underline{W}_L = (\underline{X}_{1,L})^T \underline{X}_{1,1} + (\underline{X}_{2,L})^T \underline{X}_{2,1} + \ldots + (\underline{X}_{N,L})^T \underline{X}_{N,1} \qquad (3.9)$$

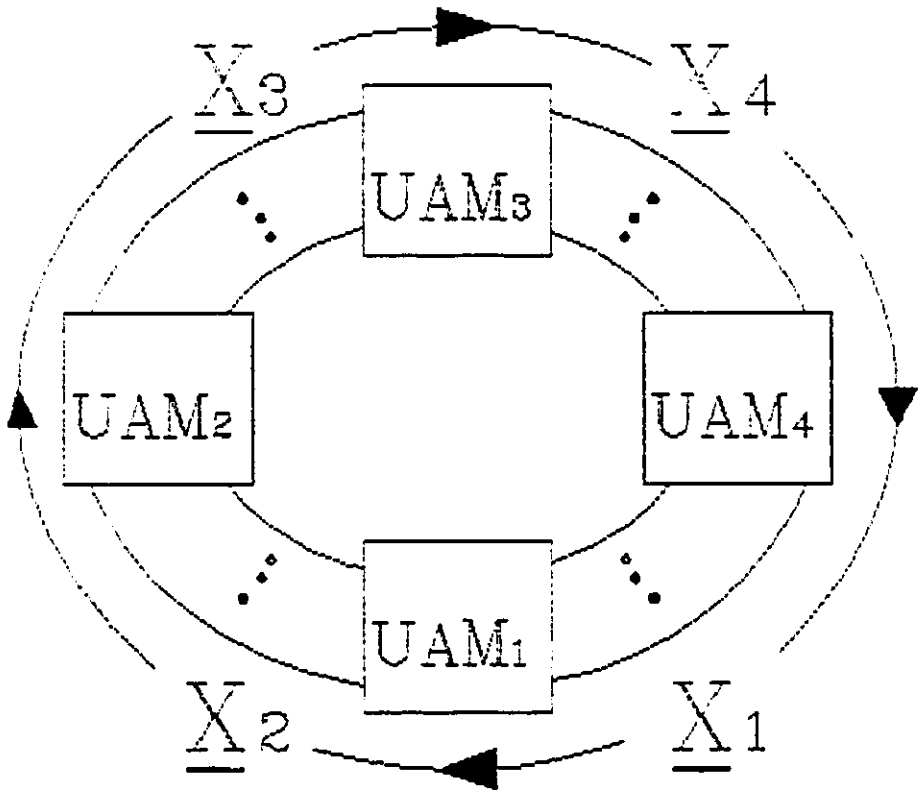


Fig. 3.3 Temporal associative memories using the RUAM.

The recalling procedure is similar to that of the CBAM but when $\underline{X}_{1,L}$ is recalled, it will recall $\underline{X}_{1,1}$ instead of passing backward to recall $\underline{X}_{1,L-1}$ as in the case of CBAM. The recalling is thus unidirectional.

This type of structure can also be recalled bidirec-

tionally by making the weight matrix identical in both directions in one AM. After recalling in one direction in one iteration, the flow direction can be reversed for next iteration.

3.2.4 Stability and Storage Capacity

Since both the CBAM and the RUAM consist of, respectively, a number of BAMs and UAMs, each of the two networks will converge to a stable state if the local energy of a BAM is decreased after one iteration and this energy is bounded. The energy of a BAM is defined as[9]

$$E_i(\underline{X}_i, \underline{X}_{i+1}) = -\underline{X}_i \underline{W}_i (\underline{X}_{i+1})^T \qquad (3.10)$$

where $\underline{X}_i$ is the input vector and $\underline{X}_{i+1}$ is the output vector.

Suppose at time k, $\underline{X}_k$ is renewed by $\underline{X}_{k-1}$ through BAM_{k-1}, the change of energy of this network will be

$$\delta E = -\underline{X}_{k-1} \underline{W}_{k-1} (\delta \underline{X}_k)^T$$
$$\leq 0$$

This inequality follows since $(\delta \underline{X}_k)^T$ and $\underline{X}_{k-1}\underline{W}_{k-1}$ always agree in sign in the corresponding matrix elements[9]. Thus, the local energy is always decreasing. Moreover, since $\underline{X}_i$ and $(\underline{X}_{i+1})^T$ are bipolar, E_i is bounded by $L(\Sigma_j \Sigma_k |W_{i,jk}|)$. Thus, all the networks discussed will converge to a stable state.

Since the CBAM and the RUAM are just constructed by connecting several BAMs, their storage capacities will be limited by that of a single BAM. However, since sequences of patterns are stored in these networks, the storage capacity will be defined as the maximum number of user-selected sequences, instead of the maximum number of data pairs as in the case of a BAM.

3.2.5 Simulation Results

A C subprogram, called TAM[21], was written to do the simulations. It is linked with three other modules (GETKEY, EDITINT, INTONE and TAM) under the project TAM.PRJ. Image sequences of four letters quantized to ±1 values were used as the temporal associative patterns in a simulation run. The letters are 54 (i.e., 6x9) pixels in size and are shown in Fig. 3.4. For the single BAM, only 2 sequences of data sets (i.e., "RING" and "WEAK") are stored as it is limited by the storage capacity. For the CBAM and the RUAM, all the four words can be stored. The pixels of the first letter which were corrupted with noise with different error probabilities, Pe (0.1, 0.3, 0.5, 0.7, 0.9), were input into the three networks to check the performances of each network. An example of each corruption is shown in Fig. 3.5.

Results are summarized in Table 1 and Table 2. In Table 1, both the CBAM and the RUAM stored only 2 sequences

of data sets (i.e., "RING" and "WEAK") so that all networks stored the same data while in Table 2, they stored all the words so that all networks reach their storage capacity limits. In Tables 1 and 2, x means incorrect recall, c means recall of the complement of the desired images, 1 means that only 1 loop ($\underline{X}_1$ -> $\underline{X}_2$ -> ... -> $\underline{X}_1$) was needed while 2 means that 2 loops were required. The results are shown in the form of the probability of occurrences for different values of Pe.

3.2.7 Comparisons

From Tables 1 and 2, we notice that all the networks performed well for low (0.1) error probability and they performed worst when Pe = 0.5. It is because at this probability of 0.5, all the stable states will have the same probability to be recalled. When Pe > 0.5, the complements of the images will most probably be recalled as the Hamming distance (the number of pixels that are different) of the corrupted input images will be closer to its complement images.

Comparing the three networks based on Table 1, we find that the single BAM is the worst in terms of correctness. It is because the storage capacity is reached in the case of the single BAM but it is not reached in the cases of the CBAM and the RUAM. From Table 1 and Table 2, we also notice

Fig. 3.5 The words used in simulation.

Pe = 0.1 Pe = 0.3 Pe = 0.5 Pe = 0.7 Pe = 0.9

Fig. 3.6 Examples of some noisy inputs of "W".

Table 3.1. Simulation Results
(All store "WEAK" and "RING")

Pe	Single BAM				CBAM				RUAM			
	1	2	c	x	1	2	c	x	1	2	c	x
0.1	.97	.03	0	0	1	0	0	0	1	0	0	0
0.3	.70	.15	0	.15	.92	0	0	.08	.89	.03	0	.08
0.5	0	.03	.22	.75	.18	0	.47	.35	.15	.03	.47	.35
0.7	0	0	.78	.22	0	0	.97	.03	0	0	.97	.03
0.9	0	0	1	0	0	0	1	0	0	0	1	0

Table 3.2. Simulation Results
(Single BAM stores two words, others store four words)

Pe	Single BAM				CBAM				RUAM			
	1	2	c	x	1	2	c	x	1	2	c	x
0.1	.97	.03	0	0	1	0	0	0	.93	.07	0	0
0.3	.70	.15	0	.15	.78	.05	0	.17	.46	.37	0	.17
0.5	0	.03	.22	.75	.08	.03	.08	.81	.03	.08	.08	.81
0.7	0	0	.78	.22	0	0	.87	.13	0	0	.87	.13
0.9	0	0	1	0	0	0	1	0	0	0	1	0

that the performances of the CBAM and the RUAM are similar but the CBAM is better in terms of the number of loop required for recall. This is because the number of weight matrix multiplications in one loop is the greatest in the CBAM (4 for the single BAM and the RUAM, but 6 for the CBAM). In fact, the use of unidirectional recalling is better than the use of bidirectional one since in unidirectional recalling, we have a more direct path, all vectors are recalled evenly and the number of summers and multipliers are much less (for storing 4 ordered vectors, the CBAM needs 6 summations of products of weights and vectors while the RUAM only needs 4, since one such summation is needed for one recall and the CBAM has 6 recalls in one loop while the RUAM has 4.).

Lastly, since different BAMs or UAMs are used to store a temporal pattern in the CBAM and the RUAM. A pattern which consists of several identical vectors can be stored in different associative memories of these two networks whereas the single BAM cannot. Moreover, the length of the temporal pattern will not be limited by the storage capacity of the BAM in the CBAM and the RUAM.

3.3 USE OF NEW ALGORITHMS

As mentioned earlier, the storage capacities of the Hopfield network and the BAM is very limited, it is essential to increase their storage capacities. The use of delta rule can improve this a lot. However, the delta rule is a slow learning process and may require many iterations in order to achieve certain convergence criteria. So if the Hebb rule can be improved, it will be best suited for associative memories for its fast and simple training.

3.3.1 Modified Hebb Rule

Since the basis of the Hebb Rule is the pairwise correlation between vector elements, this correlation is incomplete and that is why the delta rule which minimizes the error is better in storing vectors. This correlation can be improved if the effect of the noise term, which is the correlation between the different vector elements, is small compared to the correlation between the elements in one vector. For auto-associative binary data, this can be done by the introduction of a parameter in the dot product as follows:

$$w_{ij} = \sum_{k=1}^{M} \alpha_k x_i[k] x_j[k] \qquad (3.11)$$

where w_{ij} represents the weight connecting the neuron j to the neuron i of the network, $x_i[k]$ represents the ith ele-

ment of kth data to be stored, M represents the total number of sets of data to be stored, and α_k is the multiplication factor for the kth data.

α_k is to be determined by the number of '1's in the kth data and it is given by equation (3.12):

$$\alpha_1 * l_1 = \alpha_2 * l_2 = \ldots = \alpha_k * l_k = \ldots = \alpha_M * l_M = L \qquad (3.12)$$

where l_k represents the number of '1' in the kth data and L represents the L.C.M. (Least Common Multiple) of (l_1, l_2, l_3, ... l_M). Using this equation, the effect of '1', and hence the correlation, will be balanced although the number of '1' in the original vectors are different.

In this learning rule, we must use a non-linear function in recalling. An example is the step function with a threshold value. This threshold value can be found by searching from 1 to L.

A program called TESTNEW1.C[21] was written and was run in the VAX/VMS system. This program is to test the percentage of successful storage for vectors that fixed in length. In the test, the vector length was 5. All the possible combinations of vectors were generated and stored in a group of 2, 3, 4 and 5 vectors. For example, in a group of 2 vectors, there were altogether 31*30/2! = 465 combinations.

In the program, the L.C.M. is found by the following algorithm. First the H.C.F. (Highest Common Factor) of l_1 and l_2 is found using Euclidean Algorithm. Then the L.C.M. of this two number will be given by equation (3.13):

$$\text{L.C.M.} = (l_1 * l_2) / \text{H.C.F.} \qquad (3.13)$$

The values of l_1 and l_2 are updated to the L.C.M. value. The H.C.F. and hence the L.C.M. of the new l_2 and l_3 are then found and l_3 is updated. This process is continued until the last L.C.M., which is the L.C.M. of all l_k, is found. The factor α_k is also found at the same time.

The threshold values of all the neurons are found by testing whether the value can distinguish all the vectors. If there exists such a value, that set of vectors is said to be successfully stored; otherwise, the counter of incorrect storage will be increased by one.

Another program, which is called TESTHEBB.C[21] for testing the Hopfield network was also written so that a comparison can be made. The results are shown in Table 3.3. As shown in Table 3.3, it was found that the modified Hebb rule is much better than the Hebb rule in storing the vectors although the percentage of successful storage is decreased with the increase in the number of vector stored. Results show that the Hebb rule can hardly store more than 2 vectors at the same time while the modified rule can have

about 87% confidence in storing five 5-bit vectors at the same time.

Table 3.3
Comparison between the Hebb Rule and the Modified Hebb Rule

No. of vector Stored	Percentage of successful storage	
	Modified Hebb Rule	Hebb Rule
2	100.00 %	76.29 %
3	99.44 %	15.47 %
4	95.82 %	1.89 %
5	87.29 %	0.02 %

3.3.2 More Improvements of the Modified Hebb Rule

Although this algorithm can be used for recalling data in most cases, there may be some data sets which cannot be recalled completely. In this situation, there are two algorithms which can be used after the training obtained in equation (3.11).

In algorithm 1, which can be used in short-bit-length data, we can add a positive feedback into each neuron. The ith feedback has a value determined by the total number of '1' in the ith data bit of all the M sets of data and is given by the following equation:

$$f_i = B - b_i \tag{3.14}$$

where f_i is the feedback value; b_i is the total number of

'1' in the ith data bit of all the M data sets; and B is the maximum value among (b_1, b_2, ..., b_N).

In algorithm 2, which can be used in long-bit length data, we can use the following algorithm to modify the trained weight when there is any error:

Step 1a: If the output vector element is 0 in which it should be 1, add 1 to all non-decreasing weights (i.e., weights which have not been decreased in the previous iteration) connecting to that corresponding neuron.

Step 1b: If the output vector element is 1 in which it should be 0, subtract 1 from all non-increasing weights (i.e., weights which have not been increased in the previous iteration) connecting to that corresponding neuron.

Step 2: Repeat Step 1 for all other elements.

Step 3: Repeat the above steps for other vector until all training data have been used.

Step 4: Repeat the above steps if there is any error and if there is any weight which is still changing.

In this algorithm, the weights are adjusted to minimize the error. As we just modifv the trained weights, the convergence of this algorithm can be very fast and this has been verified in the simulation results.

Simulations of the network has been done for 4-bit and 5-bit vector using the algorithm 1 by using the program TESTNEW2.C[21]. All the possible input combinations were used as the input. The simulation results showed that any number of sets of vector can be stored and recalled correctly using this algorithm. 2^4 and 2^5 data had been stored at the same time and recalled correctly for 4-bit and 5-bit data respectively. Results indicate that we can have a network of full storage capacity.

For the algorithm 2, a simulation subprogram, which is called NEW1.C[21], was written. This subprogram is similar to the TAM.C and is used, in an IBM, or compatible PC, together with three other modules (GETKEY, EDITINT and INTONE). They are under the project NEW1.PRJ. Another subprogram, which is called STANDONE.C[21], was also written for the standard Hebb rule. The program is in project STANDONE.PRJ which contains the three modules and this subprogram.

Fig. 3.6 shows the 10 images that were stored in a 5x7 network. The recalled images of a network using the standard Hebb rule are shown in Fig. 3.7. It was found that all images can be recalled correctly using the new algorithm. On the other hand, no image can be recalled correctly using the standard Hebb rule.

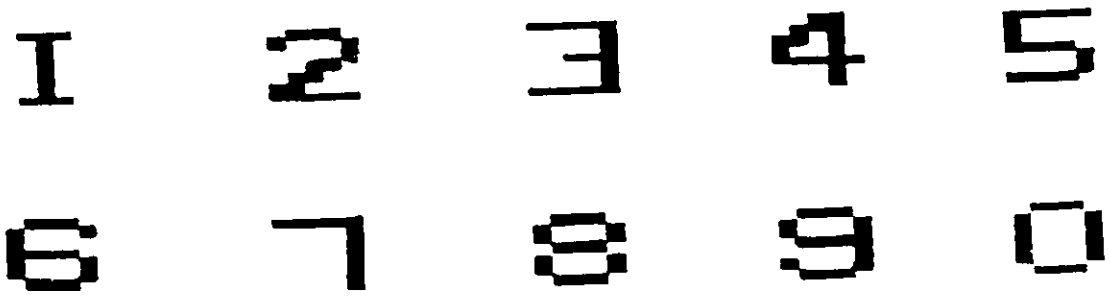


Fig. 3.6 Input 5x7 images.

Simulations were also carried out to test the usefulness of the equation (3.11) on the speed of convergence of the weights if the algorithm 2 is used. The results show that for the 5x7 images, the number of iterations for training the weights was 3 if we compute the weights first by the equation (3.11) and it was 8 if only the algorithm 2 was used. This shows that the equation (3.11) is useful for improving the convergence speed for training the weights.

3.3.3 Modified Hebb Rule for Hetero-associative Data

In the above sections, the auto-associative data is used as the stored data. The modified Hebb rule can be changed accordingly for storing hetero-associative data in BAM as in equations (3.15) and (3.16).

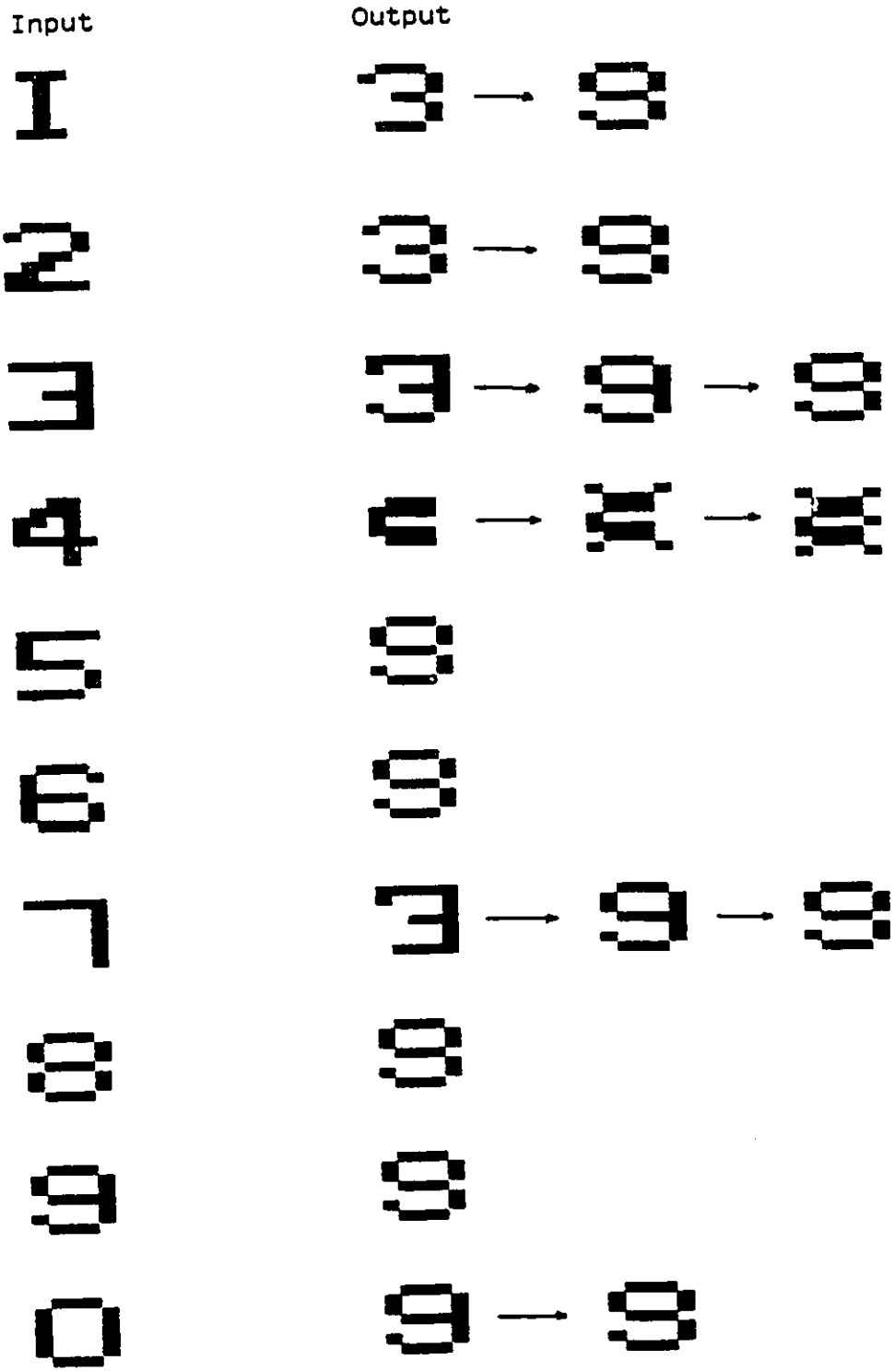


Fig. 3.8 Recalled images using the standard Hebb rule.

$$w_{ij}^{f} = \sum_{k=1}^{M} \alpha_k x_i[k] y_j[k] \qquad (3.15)$$

$$w_{ji}^{b} = \sum_{k=1}^{M} \beta_k y_j[k] x_i[k] \qquad (3.16)$$

where w_{ij}^{f} represents the weight connecting the neuron i in the layer for vector $\underline{x}$ to the neuron j in the layer for $\underline{y}$ and w_{ji}^{b} represents the opposite weight, i.e., from neuron j to neuron i. As before, $x_i[k]$ and $y_j[k]$ represent the ith element and jth element of the kth data pair to be stored respectively, M represents the total number of sets of data to be stored, and α_k and β_k are the multiplication factors for the kth data. They are determined by the equations (3.17) and (3.18) respectively.

$$\alpha_1 * yl_1 = \alpha_2 * yl_2 = \ldots = \alpha_k * yl_k = \ldots = \alpha_M * yl_M = Ly \qquad (3.17)$$

$$\beta_1 * xl_1 = \beta_2 * xl_2 = \ldots = \beta_k * xl_k = \ldots = \beta_M * xl_M = Lx \qquad (3.18)$$

where xl_k and yl_k represent the numbers of '1' in the $\underline{x}[k]$ and $\underline{y}[k]$ respectively; Lx and Ly represent the L.C.M. of $(xl_1, xl_2, xl_3, \ldots, xl_M)$ and $(yl_1, yl_2, yl_3, \ldots, yl_M)$ respectively.

3.3.4 Use of Multi-threshold Values

Although the above study shows that the storage capacity of a single layer network can be improved considerably using the modified Hebb rule, it is proved, by Minsky and Papert[14], that if the input set is not linearly separable, there is no set of weights that can provide the

required outputs in a single layer linear network. Linear separable means that given the space of all inputs, all of those which have an output one must lie to one side of a hyperplane, and all of those which have an output 0 (or -1) must lie on the other side of the hyperplane. One of the interesting problems is the XOR problem.

Consider a neuron with a binary threshold function that has two inputs. The output is described in equation (3.19).

$$X_{out} = \begin{cases} 1 \text{ if } (w_1 * x_1 + w_2 * x_2) \geq T \\ 0 \text{ if } (w_1 * x_1 + w_2 * x_2) < T \end{cases} \qquad (3.19)$$

The inputs to the neuron which correspond to the transition point of the output satisfy the equation :

$$w_1 * x_1 + w_2 * x_2 - T = 0 \qquad (3.20)$$

Equation (3.20) defines the decision boundary between inputs which have an output of 1 and those which have an output of 0 and is the hyperplane. Thus, a neuron with binary threshold function can only linearly separate a set of inputs. Since the XOR problem is not linearly separable, it cannot be solved by a neuron with binary threshold function. In order to solve this problem, a multi-threshold function can be used instead. The idea is to introduce more than one hyperplane to distinguish the inputs. Thus, any kind of problem can be solved when the number of hyperplanes is not fixed. To solve the XOR problem, the following values and threshold function can be used:

$$w1 = w2 = 1$$

$$X_{out} = \begin{cases} 0 \text{ if } & (w_1 * x_1 + w_2 * x_2) < 1 \\ 1 \text{ if } & 1 \leq (w_1 * x_1 + w_2 * x_2) < 2 \\ 0 \text{ if } & 2 \leq (w_1 * x_1 + w_2 * x_2) \end{cases} \qquad (3.21)$$

The algorithm for finding the threshold values, after finding the weights, is as follows:

1. Set the number of threshold values in one neuron to the number of patterns stored. Initialize the threshold values to zero except the one in pattern 0 which is set to maximum integer.

2. Present pattern 0 to the network and for each output neuron, calculate the total activation, called SUM1, of the neuron before applying the threshold function. Search the list of threshold values until SUM1 is smaller than or equal to a threshold value. Record the position j.

3a. If SUM1 equals the threshold value but the target output is not the same as the output associated with that threshold value, it means that there is a conflict between this pattern pair and the previous pattern pair. For example, if SUM1 is 5 and the output should be quantized to 1 for the previous pattern pair and now SUM1 is also 5 but the output should be quantized to 0, there exists a conflict. In this case, all the weights, which are connected to this neuron

from the neurons that have an output state '1' in this input pattern, will be increased by 1. This makes the SUM1 different and thus resolves the conflict. After this increment of the weights, return to step 2.

3b. Otherwise, insert SUM1 as the threshold value and the target output as the associated output to the current position j in the list of threshold values.

4. Repeat 2 with other pattern pairs until all patterns are presented.

5. Take the mean of two successive threshold values as the final threshold value and set the first threshold value to the minimum integer value. Then a list of threshold values in ascending order can be made and each threshold value will divide two patterns equally.

A subprogram, which is called NEW2.C[21], was written. It runs in an IBM, or compatible PC and is used together with the editor and getkey modules. This subprogram is linked with GETKEY, EDITINT and INTMUL in NEW2.PRJ. The modified Hebb rule without the algorithm 2 and the multi-threshold values are used. Again ten images were stored but this time, they are in pattern pairs as shown in Fig. 3.8. This network can be also used to store the images in Fig.

3.6.

Another subprogram, which is called STANDMUL.C[21] and is similar to the above subprogram, was written. It is linked with GETKEY, EDITINT and INTMUL in STANDMUL.PRJ. This subprogram implements the standard Hebb rule with multi-threshold values. It can also store the images in Fig. 3.6 and those in Fig. 3.8.

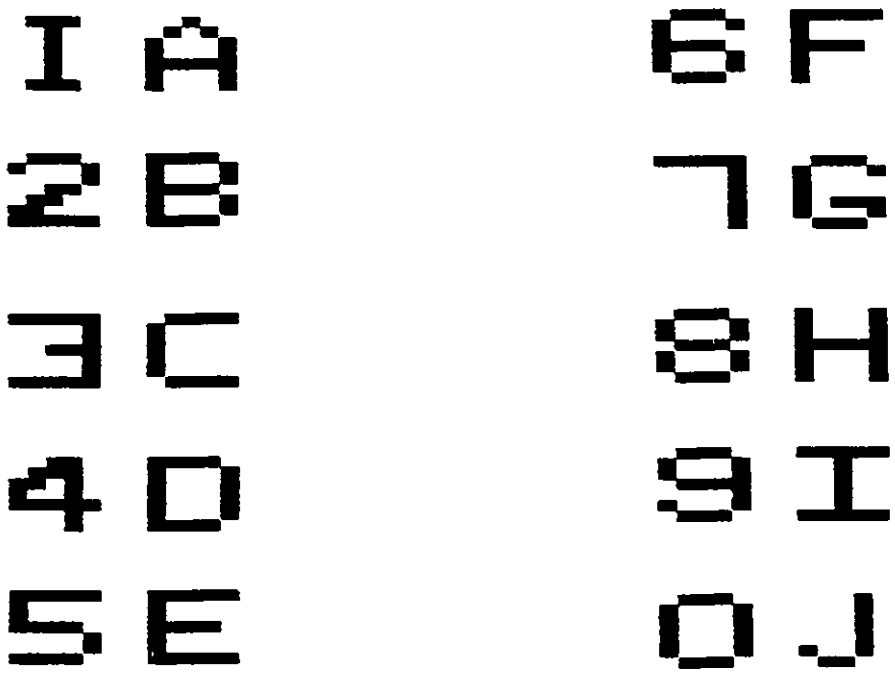


Fig. 3.8 Input 5x7 pattern pairs

3.3.5 Delta Rule with Hardlimiting Function

The delta rule can be used to store binary or bipolar associative data. There are two methods to do this. One is

without the hardlimiting function in training but with the hardlimiting function in recalling. The hardlimiting function in recalling is essential as the output image should be quantized to binary or bipolar values. The second method is with the hardlimiting function in both training and recalling.

Suppose $(\underline{X}^{[k]}, \underline{Y}^{[k]})$ are to be stored for k = 1 ,..., M. Without the hardlimiting function, the training for BAM is as follows: Initialize the weights to zero (or random values) and present each pattern pair to the network. Update the weights by the following equations in the order shown below:

$$y'_j = \sum_i w_{fij} x_i^{[k]} \quad \text{for all } j \qquad (3.22)$$

$$error_j = y_j^{[k]} - y'_j \quad \text{for all } j \qquad (3.23)$$

$$w_{fij} = lrate * error_j * x_i^{[k]} \quad \text{for all } i, j \qquad (3.24)$$

$$x'_j = \sum_i w_{bij} y_i^{[k]} \quad \text{for all } j \qquad (3.25)$$

$$error_j = x_j^{[k]} - x'_j \quad \text{for all } j \qquad (3.26)$$

$$w_{bij} = lrate * error_j * y_i^{[k]} \quad \text{for all } i, j \qquad (3.27)$$

The training is continued until certain convergent criteria are met. The recalling is performed by equations (3.22) and (3.25) but this time they should be quantized to (1, -1) for bipolar data and (1, 0) for binary data using threshold function.

With the hardlimiting function in training, y'_j in equation (3.22) and x'_j in equation (3.25) will pass through the hardlimiting function (threshold value = 0) before computing the errors. The rest is the same as above.

Simulation subprograms were written and are similar to the subprograms discussed before. DELTA1[21] implements the first method, i.e., without the hardlimiting function in training, while DELTA2[21] implements the second method. DELTA1 is linked with GETKEY, EDITFLOAT, FLOATONE in DELTA1.PRJ while DELTA2 is linked with the same modules in DELTA2.PRJ. All these networks can store the images in Fig. 3.6 and Fig. 3.8.

3.3.6 Comparisons

To sum up this topic, a comparison is made with all the methods discussed above. The images in Fig. 3.6 and Fig. 3.8 are used as the auto-associative and the hetero-associative data respectively. The results are shown in Table 3.4, Table 3.5, Table 3.6 and Table 3.7. In Table 3.4 and Table 3.5, all the ten images (pairs) are stored while in Table 3.6 and Table 3.7, only the first three images are stored. The reason for using 3 images is that it is approximately the storage capacity (using $n/2\log_2(n)$ [12]) of the standard Hebb rule with hardlimiting function. Using Table 3.6 and Table 3.7, the performances of the networks

for storing few (compared to maximum storage capacity) data can be compared. In the tables, the number is the probability of successful recalling when 50 trials were done. The values under the column Pe is the probability error which is the probability that each pixel reverses its state. Different rules are represented by the program names: STANDONE uses the standard Hebb rule, STANDMUL uses the standard Hebb rule with multi-threshold values, NEW1 uses the modified Hebb rule with algorithm 2 applied (section 3.3.2), NEW2 uses the modified Hebb rule with multi-threshold values, DELTA1 uses the delta rule without the hardlimiting function in training, and DELTA2 uses the delta rule with the hardlimiting function (In using the delta rule, convergence criteria that the total square error is smaller than 0.1 and the learning rate parameter 0.05 are used). The number in brackets shows the training time required for each algorithm. The computer used was an IBM AT of 10MHz clock rate with mathematics coprocessor 80287.

Based on the tables, the following observations can be made:

1. Comparing the training time, the standard Hebb rule is the fastest since there is no iteration in the training. The delta rule without the hardlimiting function in the training is the slowest because the output has to be close enough to the target in order to stop the training and this

Table 3.4 Performances for Storing 10 Auto-associative Data

Pe	STANDONE (1s)	STANDMUL (2s)	NEW1 (110s)	NEW2 (3s)	DELTA1 (307s)	DELTA2 (20s)
0.0	0	1	1	1	1	1
0.1	-	0.38	0.14	0.34	0.42	0.34
0.3	-	0.10	0.06	0.02	0.00	0.00

Table 3.5 Performances for Storing 10 Hetero-associative Data

Pe	STANDONE (1s)	STANDMUL (21s)	NEW1 (-)	NEW2 (3s)	DELTA1 (455s)	DELTA2 (50s)
0.0	0	1	0	1	1	1
0.1	-	0.02	-	0.00	0.10	0.34
0.3	-	0.00	-	0.00	0.04	0.20

Table 3.6 Performances for Storing 3 Auto-associative Data

Pe	STANDONE (1s)	STANDMUL (1s)	NEW1 (2s)	NEW2 (1s)	DELTA1 (25s)	DELTA2 (4s)
0.0	1	1	1	1	1	1
0.1	0.86	0.96	0.52	1.00	0.84	0.88
0.3	0.48	0.44	0.36	0.58	0.04	0.62

Table 3.7 Performances for Storing 3 Hetero-associative Data

Pe	STANDONE (1s)	STANDMUL (1s)	NEW1 (3s)	NEW2 (1s)	DELTA1 (25s)	DELTA2 (4s)
0.0	1	1	1	1	1	1
0.1	1.00	1.00	0.26	1.00	0.84	0.96
0.3	0.80	0.72	0.10	0.90	0.14	0.62

needs many iterations. Comparing the modified Hebb rule with multi-threshold values to the standard Hebb rule with multi-threshold values, it is shown that the modified Hebb rule is faster and thus is useful for storing more hetero-associative data.

2. When storing 10 auto-associative data, the delta rule without the hardlimiting function is the best for Pe=0.1. However, the performance of this rule with or without the hardlimiting function degrades a lot (cannot recognize any image) when the probability error increases to 0.3. So the standard Hebb rule with multi-threshold values, which is the best on average, should be used for this case.

3. When storing 10 hetero-associative data, the delta rule with the hardlimiting function is the best. Other methods can hardly recognize a noisy image as there are too many data stored.

4. When storing 3 auto-associative data, the modified Hebb rule with multi-threshold values is the best for Pe=0.1 and its performance is also comparable to others when probability error is increased to 0.3.

5. When storing 3 hetero-associative data, again the

modified Hebb rule with multi-threshold values is the best.

6. In general, if more data are stored, all the networks perform better for auto-associative data. On the other hand, if only a few number of data are stored, all the networks perform better for hetero-associative data.

7. The modified Hebb rule with algorithm 2 performs the worst and is unsuitable for storing Hetero-associative data.

8. A network will degrade its noise performance when more data are stored. This is reasonable as the number of stable states is increased when more data are stored. The maximum hamming distance for distinguishing two data will be decreased and thus, for a fixed probability error, more wrong recallings will occur.

To sum up, based on the time and noise performance of the networks, the modified Hebb rule with multi-threshold values should be used unless many hetero-associative data are stored. In the latter case, the delta rule with the hardlimiting function should be used.

CHAPTER IV APPLICATIONS

After discussing the storage capacities of some single layer networks, the next step for investigating the neural networks is their applications other than storing assoicative data. Since the applications using multi-layer neural network with back-propagation learning rule are well published[20], this network will be used here again for its ability for pattern classification.

4.1 USE IN PULSE COMPRESSION

4.1.1 Traditional Approaches

To improve the performance in pulse radar detection or flaw detection, pulse compression[15], which involves the transmission of a long duration wide bandwidth signal code, and the compression of this received signal to a narrow pulse, is always employed. A typical system is shown in Fig. 4.1.

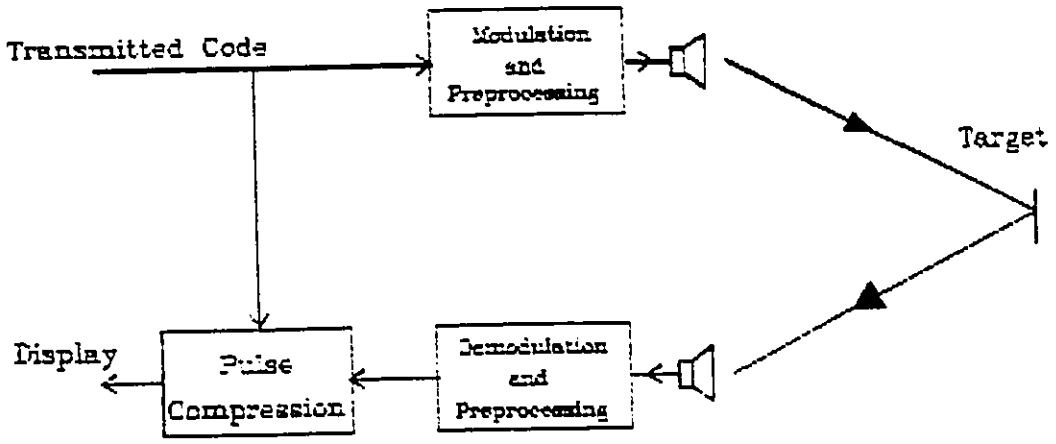


Fig. 4.1 A simplified pulse compression system.

Practically, two different approaches can be used. The first one is to use a matched filter which is actually a correlator[15] and the second one is to use mismatched filter optimum in the least-mean-square sense.

In the first approach, codes with small sidelobes in their autocorrelation functions should be used. After receiving the reflected code, the reflected code will be correlated with the transmitted code. The output will be the correlation function. Thus, if the autocorrelation function of the transmitted code is a spike function with small sidelobes, the compression will be better in that the signal, i.e. the spike, to the maximum sidelobe ratio is

larger. This signal-to-sidelobe ratio is important as the sidelobes may be regarded as a false target. So this signal-to-sidelobe ratio is a quality measure of the filter.

There are two kinds of inverse filters that can be used in the second approach. One is to use a non-recursive time-invariant causal filter[1] and the other is to use a recursive filter[13]. The idea of this approach is to find the inverse of the transmitted code in least-mean-square sense. The convolution of the code with its inverse will produce a spike function and thus a pulse compression can be achieved. Since there is no perfect inversion of a finite transmitted code, the result of this filter is just an approximation.

4.1.2 The Neural Network Approach

The codes used were the 13-element Barker code [15] which has the sequence (1, 1, 1, 1, 1, -1, -1, 1, 1, -1, 1, -1, 1) and the maximum-length sequences(m-sequences) of lengths 15, 31 and 63. (all of them are single-period)[15]. Each of the networks used has n input units, where n is the code length, 3 hidden units and 1 output unit. The number of hidden units is chosen to be 3 by pruning the neural network[18] for Barker code as follows:

1. Start with a large number (the same number of the input neurons, i.e., 13 for Barker code, was used) of the hidden neurons.

2. Train and recall the network to see whether the number of hidden neurons is enough.

3a. If it is enough, observe all the hidden neurons to see whether there are some neurons that never change their states (fire) during recalling and disable them if they exist. Also observe and disable any hidden neurons that are redundant. For example, if two hidden neurons always fire at the same time, one is redundant.

3b. If it is not enough, use more hidden neurons and repeat 2 and 3.

For the m-sequences, although more than three will be fired, it is easy to notice that three hidden neurons are sufficient to recognize the sequences. When the whole sequence is presented to the networks, not all the hidden neurons will be activated while for other situations, at least three hidden neurons will be activated. Thus, for the m-sequences, again three hidden neurons were used.

The output unit is the classifier which has a value one if the trained code is received and presented wholly in the input layer. The learning rule used here was the back-propagation (The details of this learning rule is discussed in Chapter II). The sigmoid function was used as the activation function of the neurons in the hidden and output layers and direct connection was used instead in the input

layer. A bias unit, which always has a value of 1, is connected through trainable weights to all the neurons in the hidden and the output layers to achieve a better classification. The corresponding connection weight $t_j^{[k]}$ connecting the bias unit to each of the jth neuron in the kth layer was obtained during training. The structure of the network for the Barker code of length 13 is shown in Fig. 4.2.

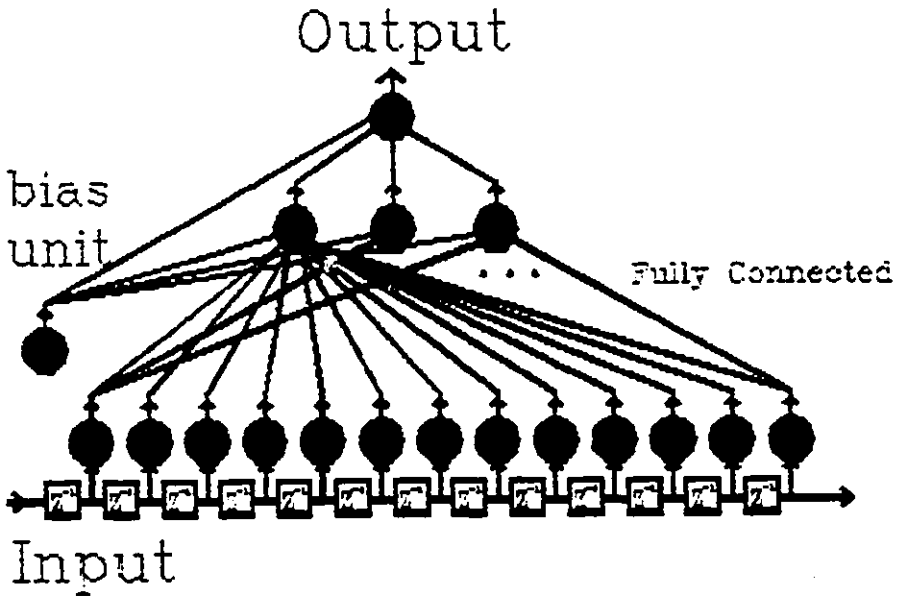


Fig 4.2 The network for pulse compression using the Barker code.

Before using this network, we should train the network by presenting a training input set repeatedly to the input layer and a corresponding target to the output layer. In each network, the training input set was just the time-shifted sequences with magnitude ±1 of the code adopted.

For example, in using the 13-element Barker code, there were 26 such sequences which included a null sequence. The desired (or target) output was 1 when the whole code was the input and it was 0, otherwise.

Initially, the values of all the weights are set randomly between ±0.1. Training is stopped by a convergence criterion which was that the maximum output rms error was less than 0.03 among all 26 possible input sequences. Output rms error of input pattern p is defined as

$$e_{rms} = \sqrt{(t_{p1} - o_{p1}[2])^2} \qquad (4.1)$$

where t_{p1} represents the target output and $o_{p1}[2]$ is the computed output at the output layer (i.e. layer 2).

The number of training epochs was found to be 500 in training the Barker code. The same number of training epochs is used in the m-sequence for comparison.

After the training, the network can be used as pulse radar detector. Received signal is clocked into the network. Thus, if the clock is fast enough for one real addition and one real multiplication which are the processing needed in each neuron, this network can be used in real-time processing.

A code generation program called CODEGEN.C[21], was

written. This program can generate four different codes: the Barker code, 15-bit, 31-bit and 63-bit m-sequences. The output codes of this program can be a noiseless code, a code with any noise, two codes with any magnitude ratio added together with any time delay, a code that misses one bit and a code that has a bit duplicated. The output is in a format that can be accessed by the software package Network Professional II of NeuralWare Incorporation. This software can implement most of the well-known neural network as required. Please refer to the user guide[20] for the details of this software. Simulations were conducted so that the noise performance of the network with different input SNR, and the resolution ability of two mixed pulse trains with different delays and input magnitude ratios were obtained. Since the received signal is clocked into the network, misalignment may occur. Simulation of this situation was also done to find out the network performance. Moreover, the effects of the number of hidden neurons on network performance, received signal magnitude on output maximum magnitude and output signal-to-sidelobe ratio, code length on output signal-to-sidelobe ratio and training epochs on output signal-to-sidelobe ratio were all found. All the results are summarized in the following two sections.

4.1.3 Network Performance

A. Noise Performance

Signals with different background noise amplitude were used as input. It was found that the output peak signal-to-sidelobe ratio were 42.73dB and 49.71dB for the Barker code and the 63-bit m-sequence respectively when no input noise was added. This is much greater than those using the traditional approaches [1, 15]. Using the Barker code, the peak signal-to-sidelobe ratio is 22.3dB if matched filter without sidelobe reduction filter is used[15] and it is 24dB if non-recursive inverse filter of length 13 is used[1]. For the 63-bit m-sequence, the peak signal-to-sidelobe ratio is about 17dB if a matched filter without a sidelobe reduction filter is used[15]. Since, in a feed-forward back-propagation neural network the error will be reduced with an increase in the number of training epochs, this ratio can be improved further before saturation is reached. When the input signal was corrupted with noise, the performance was degraded slightly. Using the 63-bit m-sequence, it could have an output peak signal-to-sidelobe ratio 25.73dB even the input SNR was 0dB. The performance of these networks for different input SNRs is plotted in Fig. 4.3. Two samples of the compressed waveforms for no added noise and 3dB input SNR, when the Barker code was used, are shown in Fig. 4.4 and Fig. 4.5 respectively. Two other samples of the compressed waveforms when the 63-bit m-sequence was used are also shown in Fig. 4.6 and Fig. 4.7.

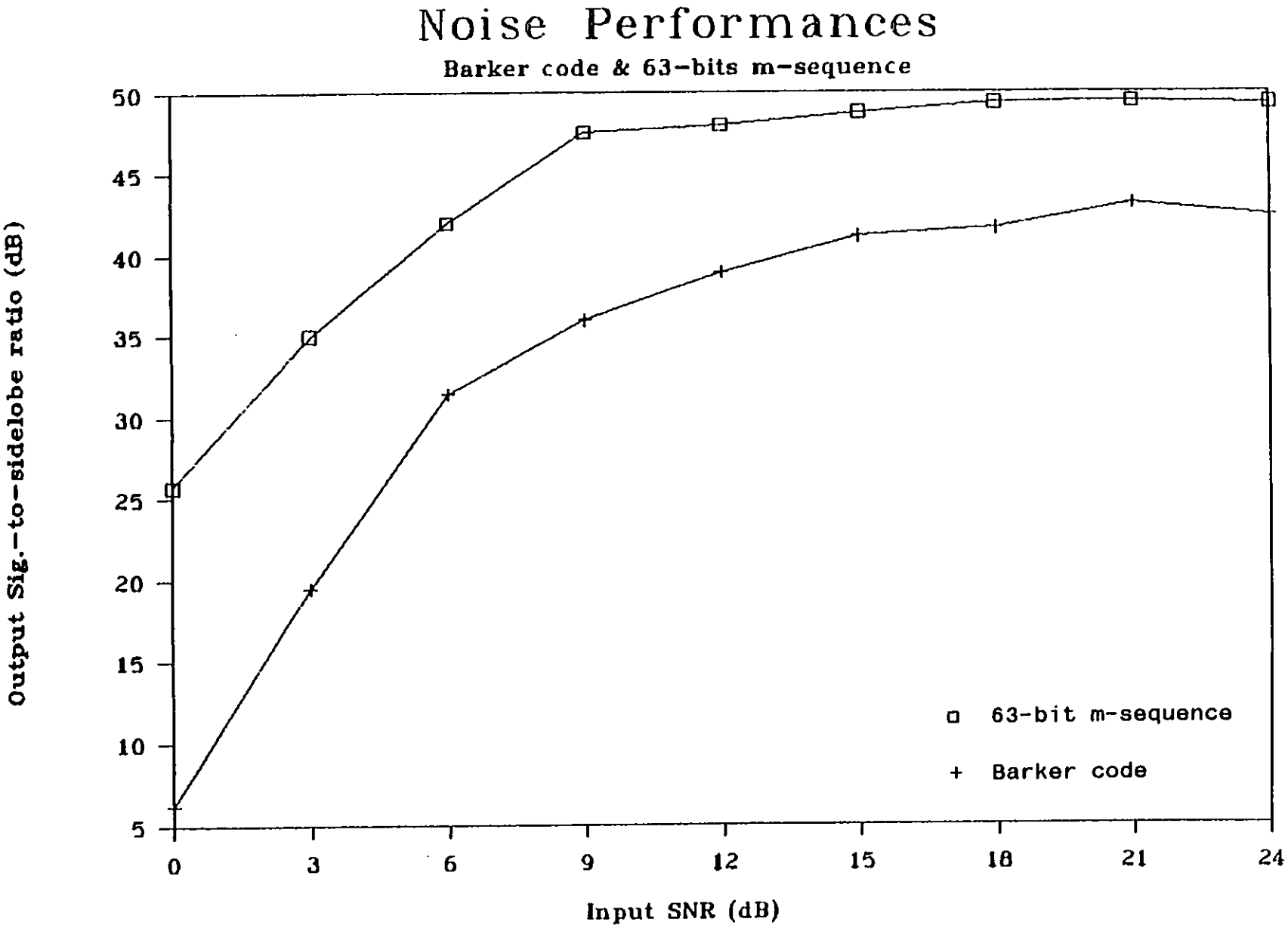


Fig. 4.3 Noise performances using Barker code and 63-bit m-sequence.

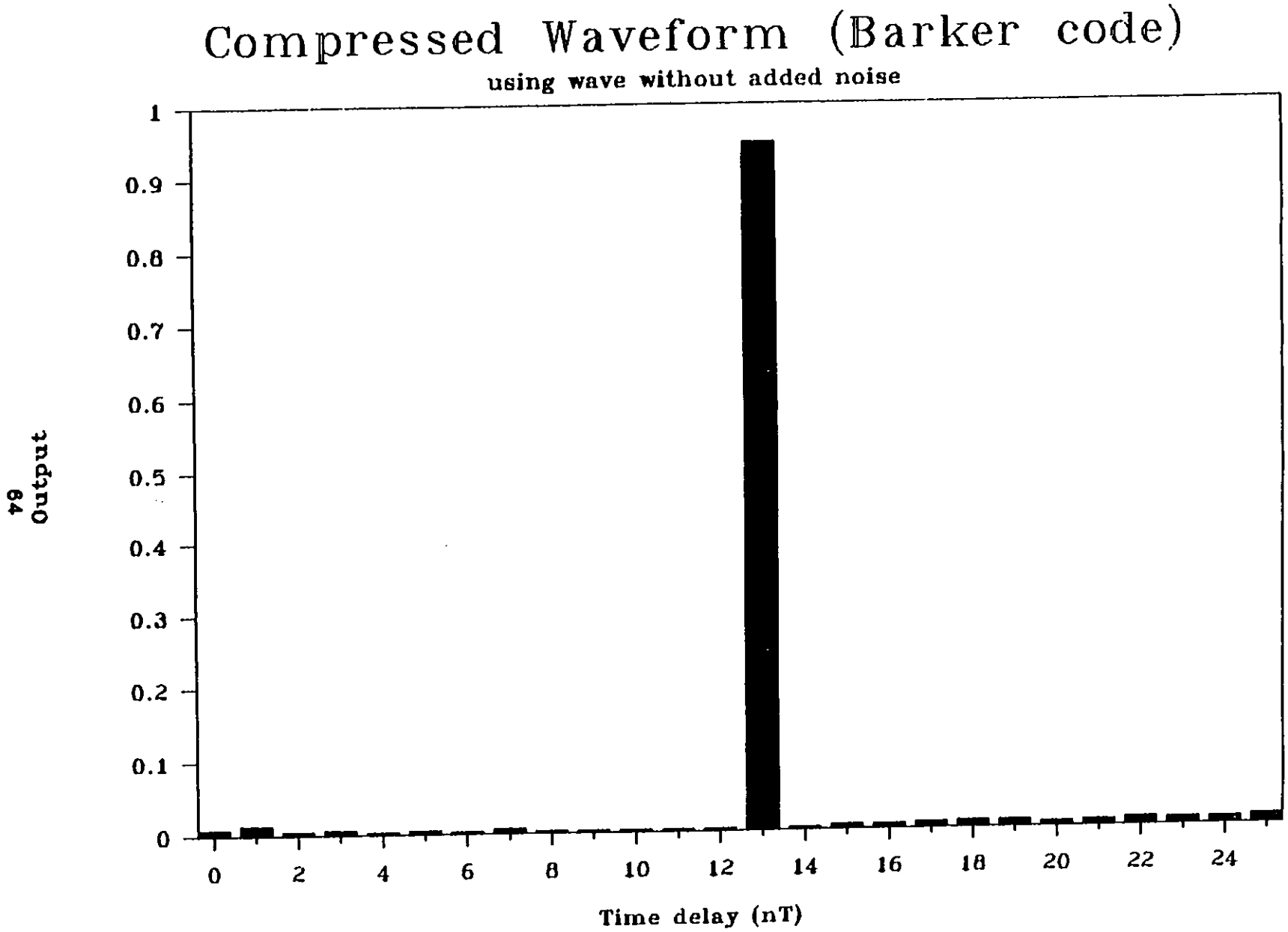


Fig. 4.4 Compressed waveform using Barker code without added noise.

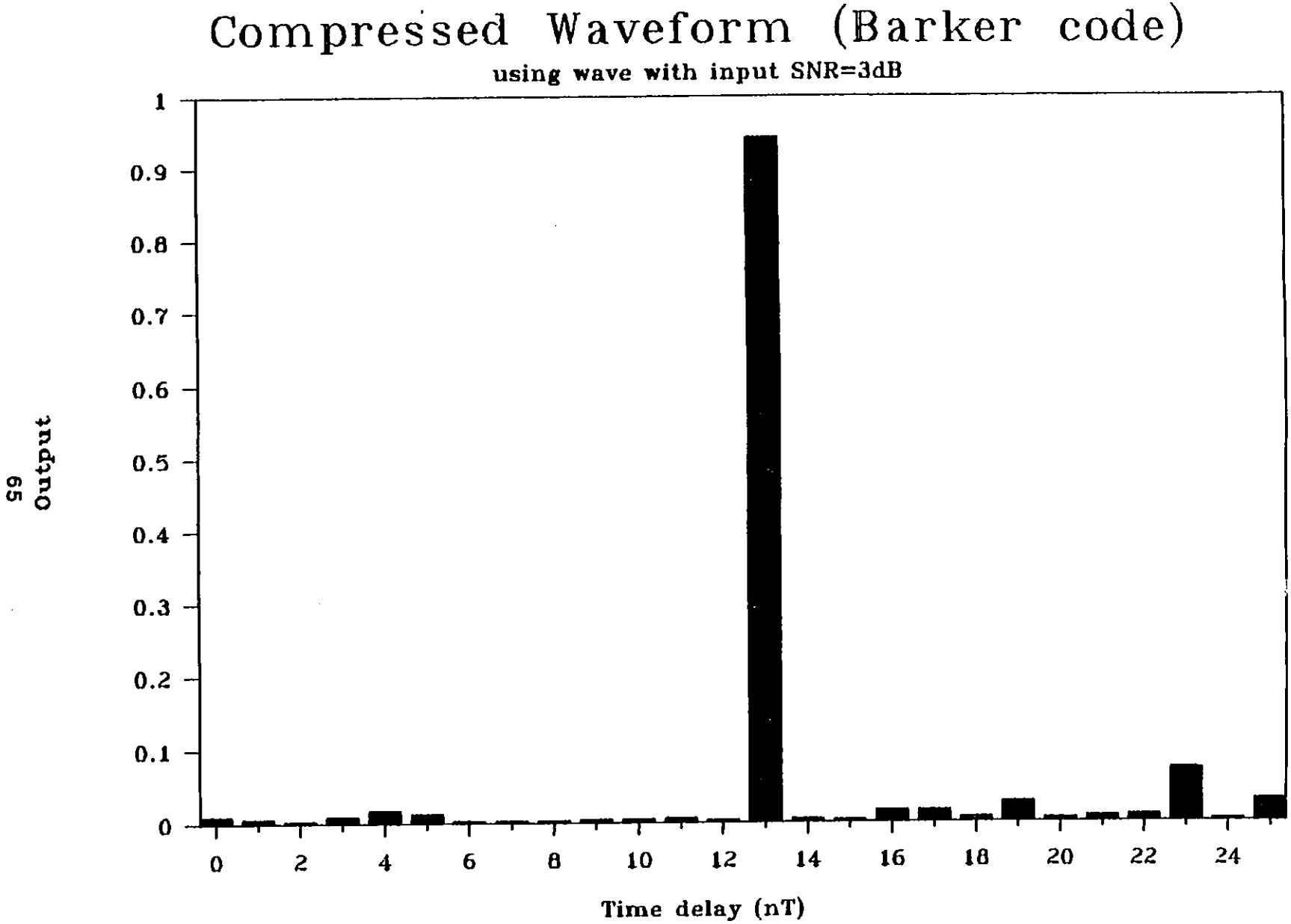


Fig. 4.5 Compressed waveform using Barker code with input SNR=3dB.

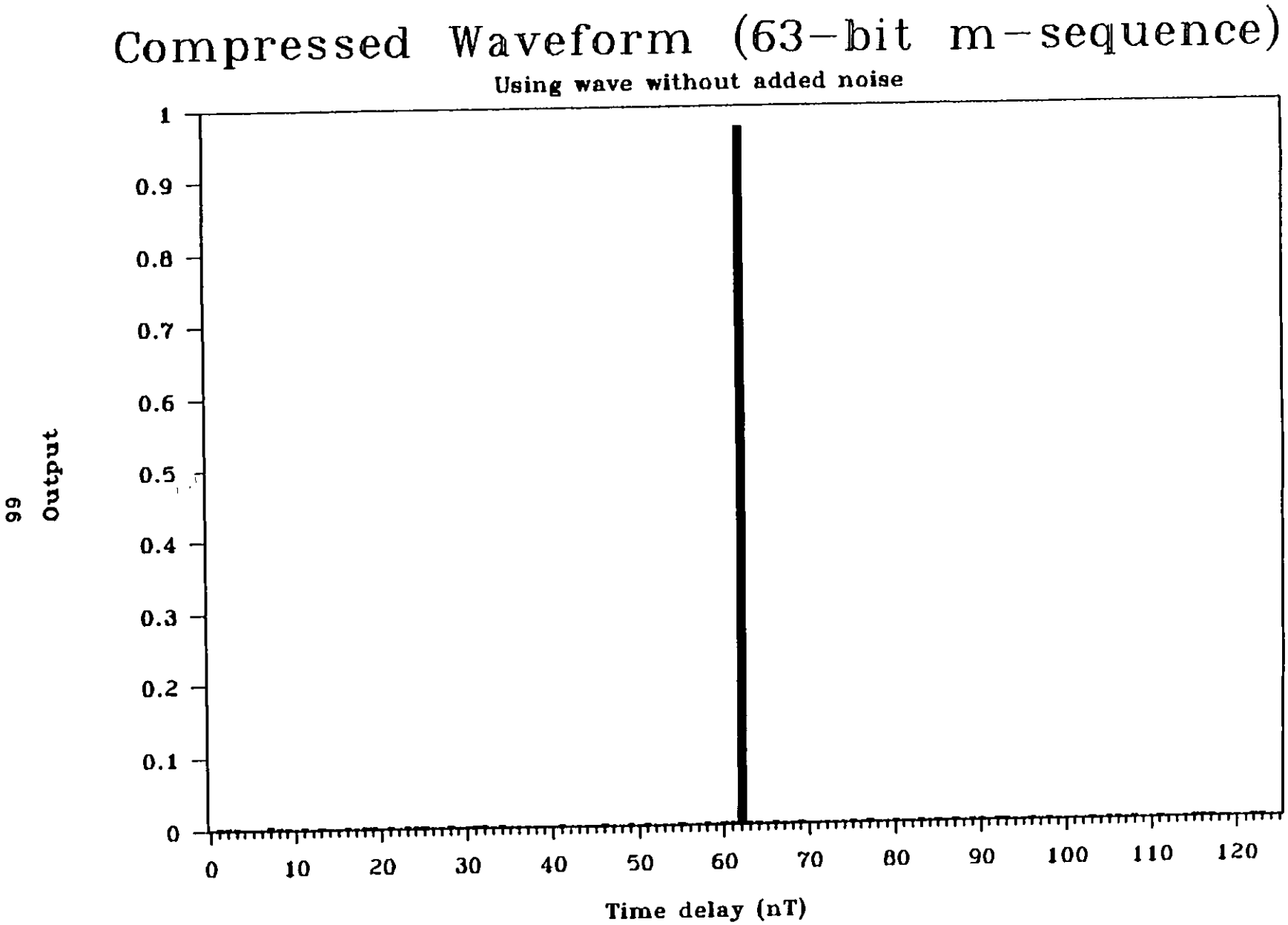


Fig. 4.6 Compressed waveform using 63-bit m-sequence without added noise.

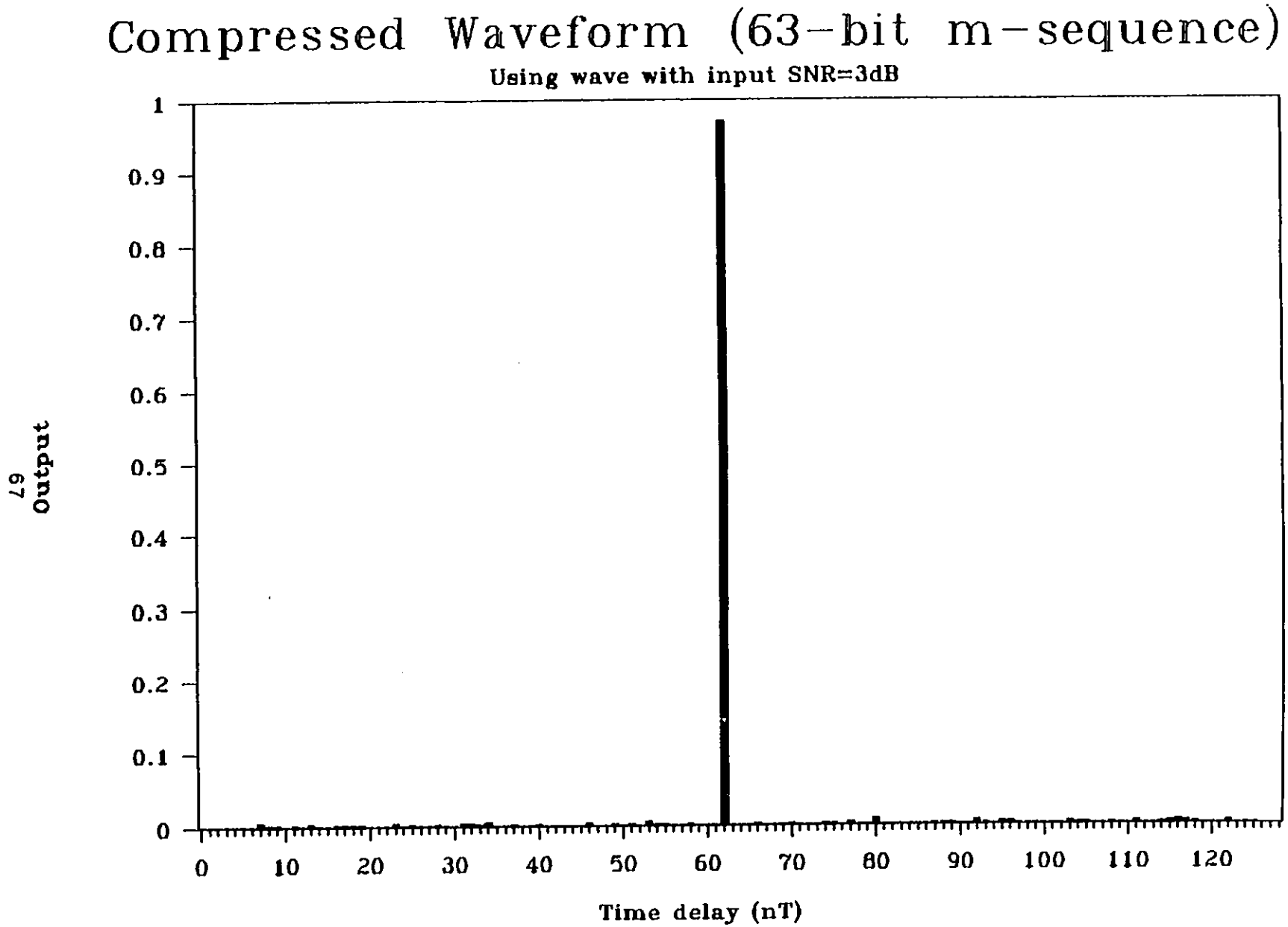


Fig. 4.7 Compressed waveform using 63-bit m-sequence with SNR=3dB.

B. Resolution Ability

Two pulse trains of the same code (using either the Barker code or the 63-bit m-sequence) with n delays apart (n=1, 3) were added and used as the input of the network (Fig. 4.8). These added pulse trains were used to simulate the receptions of 2 overlapping reflected waves. The input magnitude ratio of these two pulse trains was changed to see the effects on the performance. For the Barker code, the compressed waveforms of the added pulse trains, with three-delay-apart and equal magnitude and that with three-delay-apart and received input magnitude ratio 2 are shown in Fig. 4.9 and Fig. 4.10, respectively. The corresponding compressed waveforms, when the 63-bit m-sequence was used, are shown in Fig. 4.11 and Fig. 4.12, respectively. Other results such as the output magnitude ratio and the corresponding SNR(dB) are summarized in Table 4.1 and Table 4.2 for the Barker code and the 63-bit m-sequence respectively. In both Tables, the input magnitude ratio is defined as the magnitude of the first pulse train over that of the delayed pulse train. The output magnitude ratio is defined as the magnitude ratio of the two compressed waves. Finally, the SNR is defined as the minimum signal of the smaller compressed wave over the maximum peak noise. From Table 4.1 and Table 4.2, we found that for the Barker code, we can hardly resolve 2 signals with 1 delay apart and magnitude ratio 5 due to the small output SNR value.

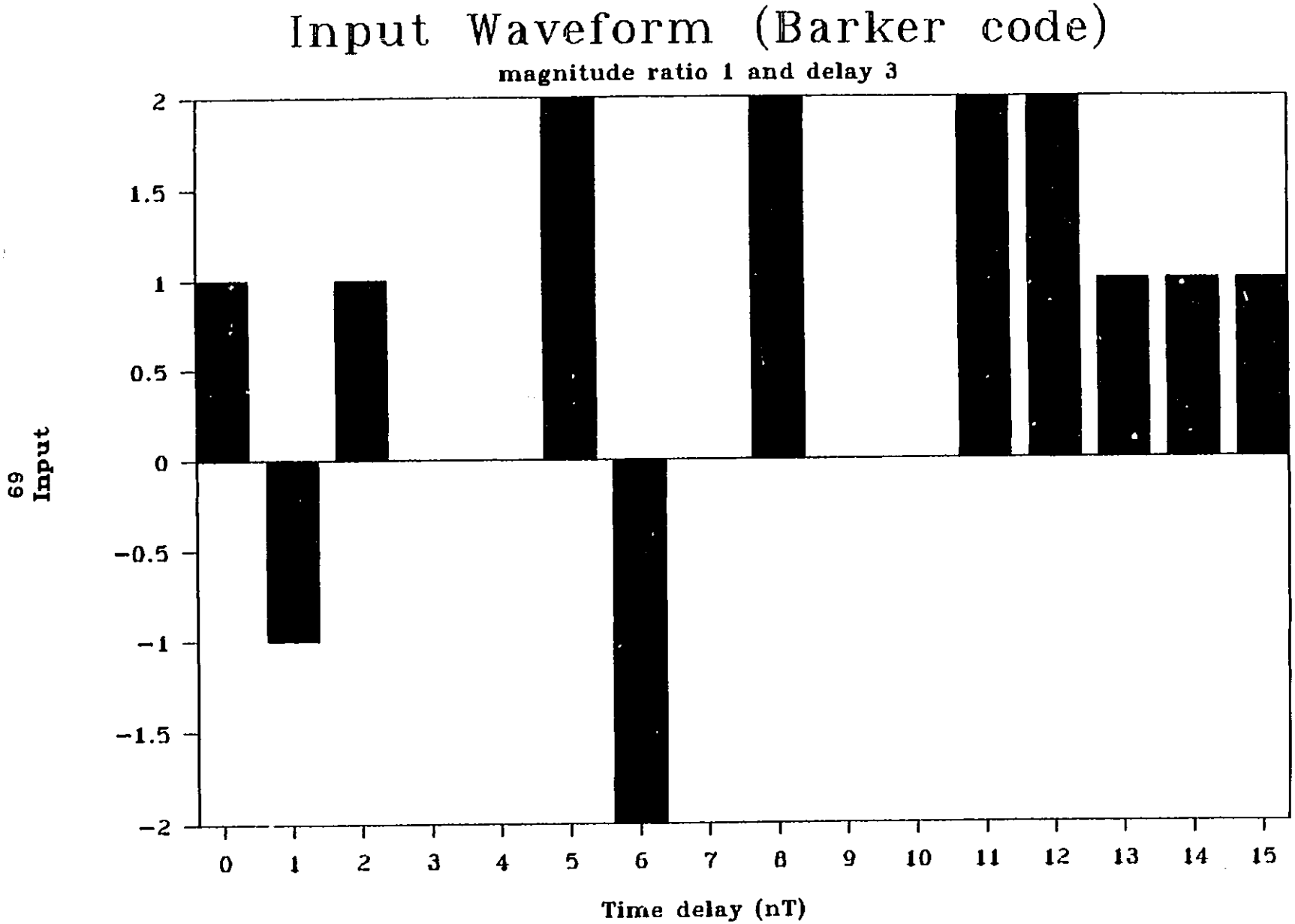


Fig. 4.8 Input waveform of two added 3-delay-apart Barker codes having same magnitude.

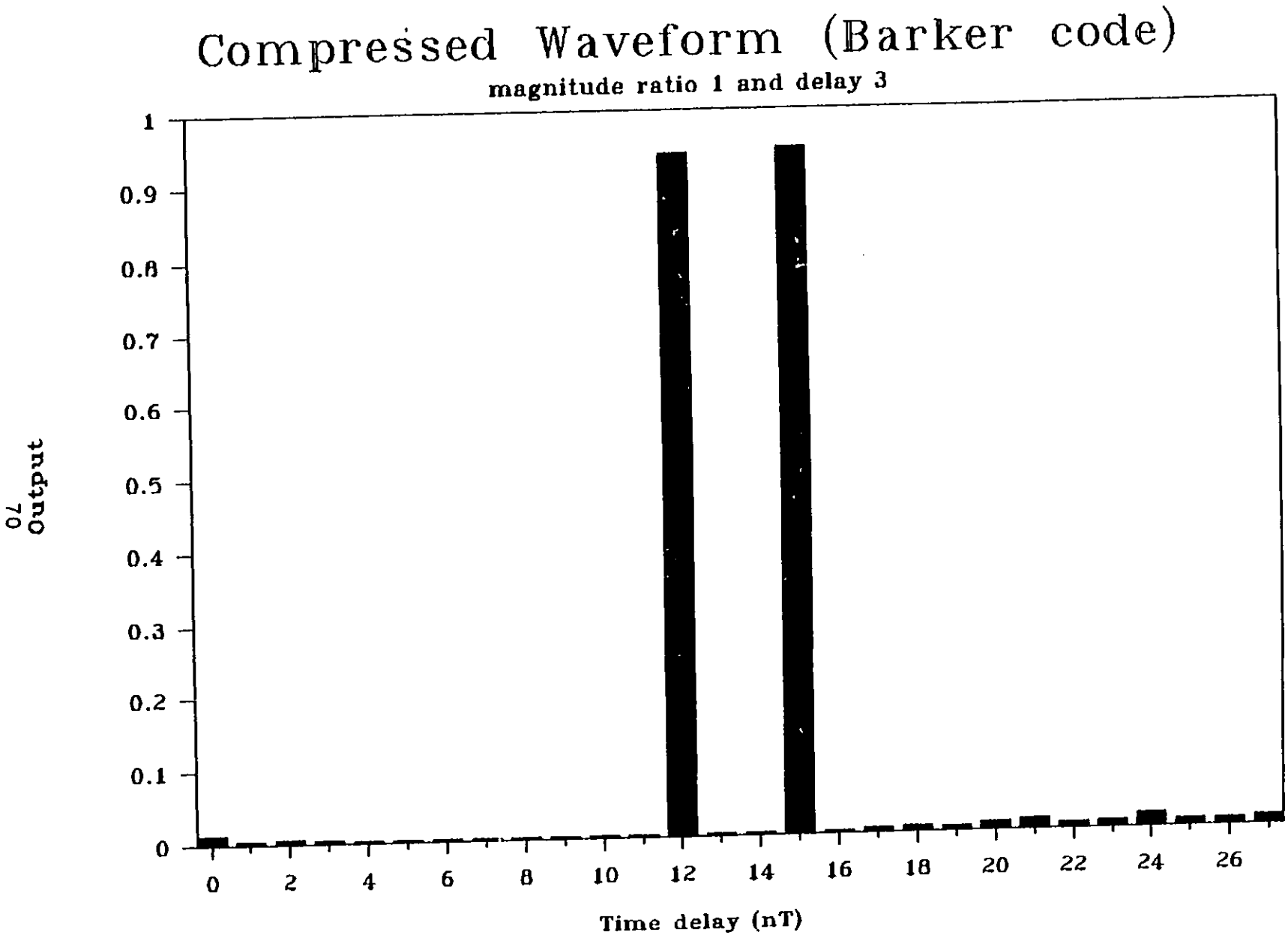


Fig. 4.9 Compressed waveform of two added 3-delay-apart Barker codes having same magnitude.

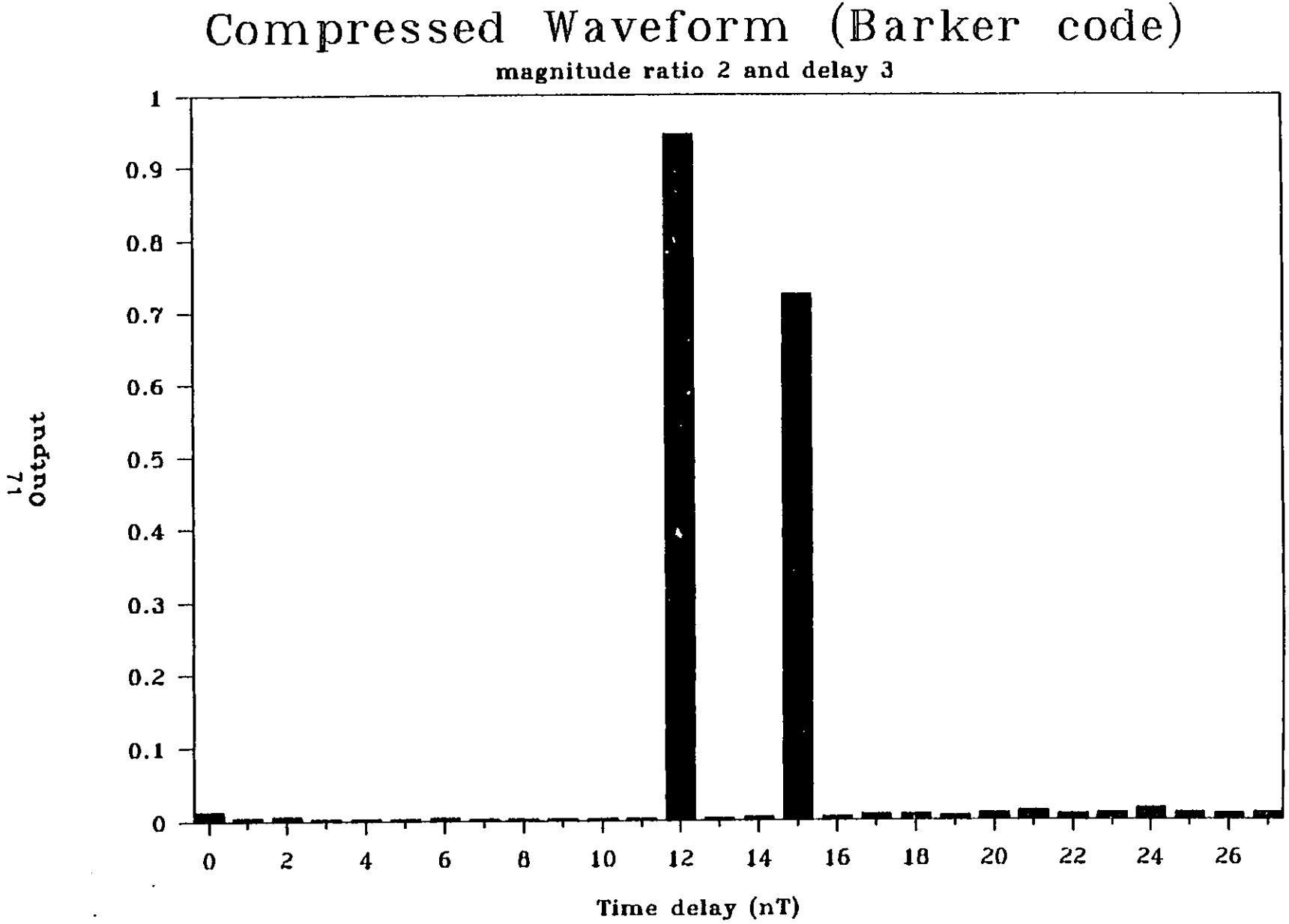


Fig. 4.10 Compressed waveform of two added 3-delay-apart Barker codes having magnitude ratio 2.

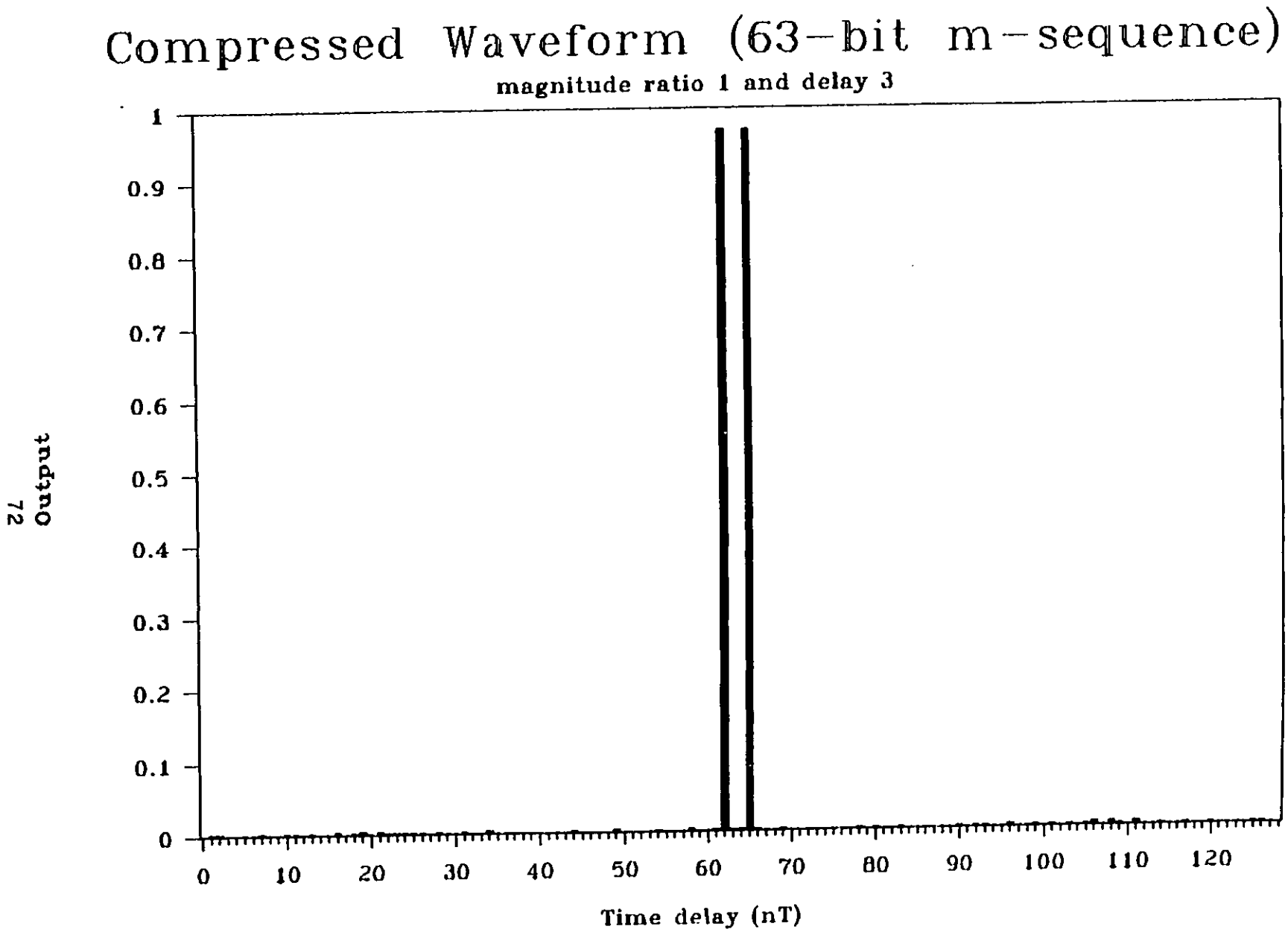


Fig. 4.11 Compressed waveform of two added 3-delay-apart 63-bit m-sequence having same magnitude.

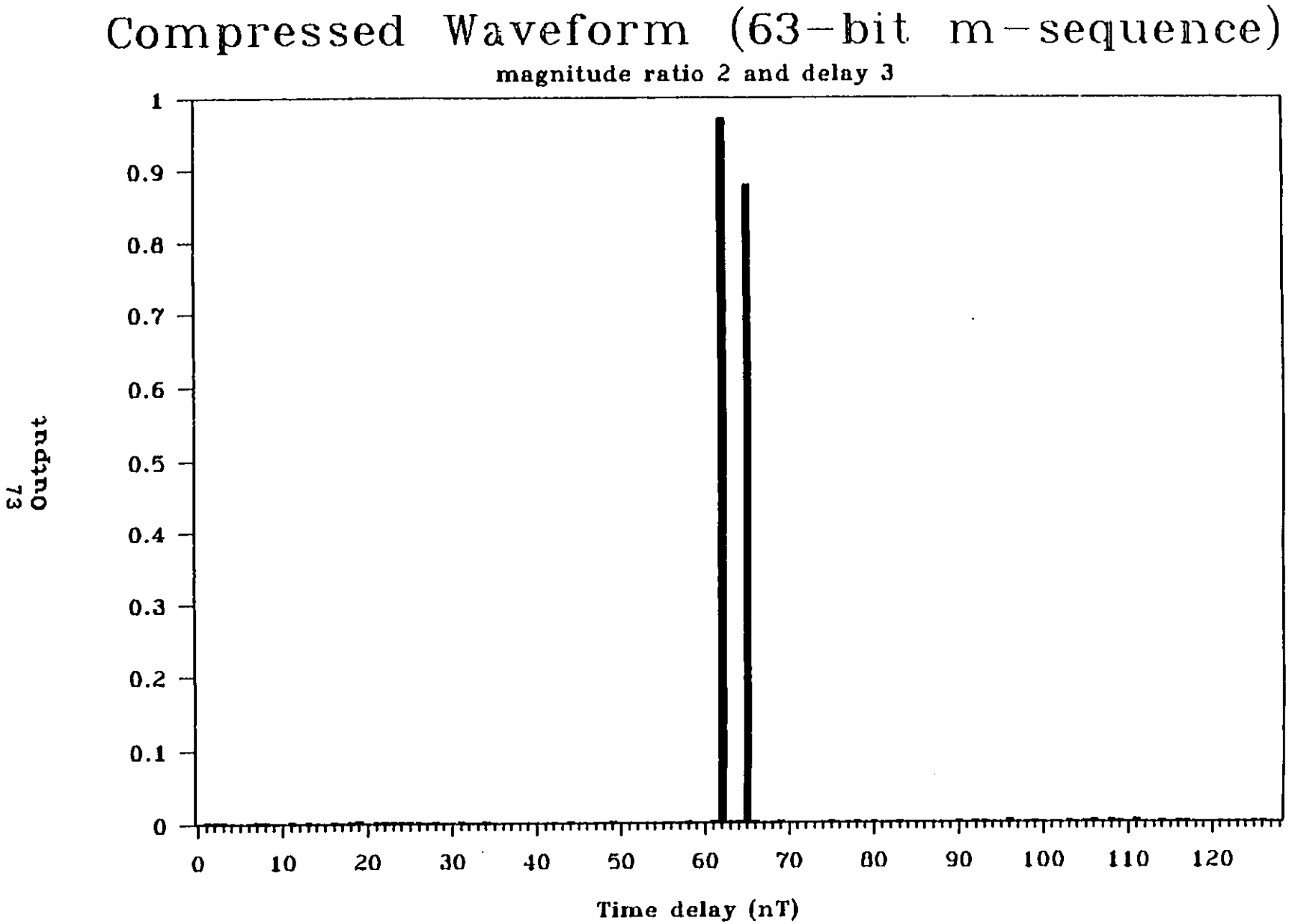


Fig. 4.12 Compressed waveform of two added 3-delay-apart 63-bit m-sequence having magnitude ratio 2.

Table 4.1 Resolution Ability for Barker Code

a. Pulse trains are 3 delays apart

Input Mag. Ratio	Output Mag. Ratio	SNR (dB)
1	1.00	38.96
2	1.24	39.00
5	28.17	13.02

b. Pulse trains are 1 delay apart

Input Mag. Ratio	Output Mag. Ratio	SNR (dB)
1	1.00	40.76
2	2.14	35.23
5	121.98	0.65

Table 4.2 Resolution Ability for 63-Bit M-Sequence

a. Pulse trains are 3 delays apart

Input Mag. Ratio	Output Mag. Ratio	SNR (dB)
1	1.00	44.94
2	1.10	47.28
5	39.26	17.99

b. Pulse trains are 1 delay apart

Input Mag. Ratio	Output Mag. Ratio	SNR (dB)
1	1.00	42.43
2	1.08	46.39
5	25.23	21.44

C. Misalignment Performance

Misalignment of clock can be simulated by duplicating (for fast clock) or discarding (for slow clock) some bits. Simulations have been done for the Barker code and the 63-bit m-sequence. The middle bit, i.e., the 7th bit in the Barker code (about 7.7% of the total number of bits) and 32th bit in the m-sequence (about 1.6% of the total number of bits), was duplicated in one simulation and the same bit was discarded in another simulation. In both cases, the network could still detect the signal. However the signal-to-sidelobe ratio is smaller. For the duplication of bit, the ratio is 21.66dB for the Barker code and 37.45dB for the m-sequence; while for the discarding of bit, the ratio is 32dB for the Barker code and 38.45dB for the m-sequence. For the m-sequence, the main-lobe width is extended to 2 delays when there is a clock misalignment (Fig. 4.13 and Fig. 4.14). These two time delays can be treated as the reception of the transmitted code with a poorer accuracy. The magnitude ratios of these two spikes and the corresponding signal-to-sidelobe ratios of other misalignment positions using the 63-bit m-sequence are shown in Table 4.3. These simulations show that the network still works for clock misalignment. In Table 4.3, mag. ratio stands for the magnitude ratio between the two successive spikes, SNR is the signal-to-sidelobe ratio with the higher spike as the signal value and highest sidelobe other than the lower

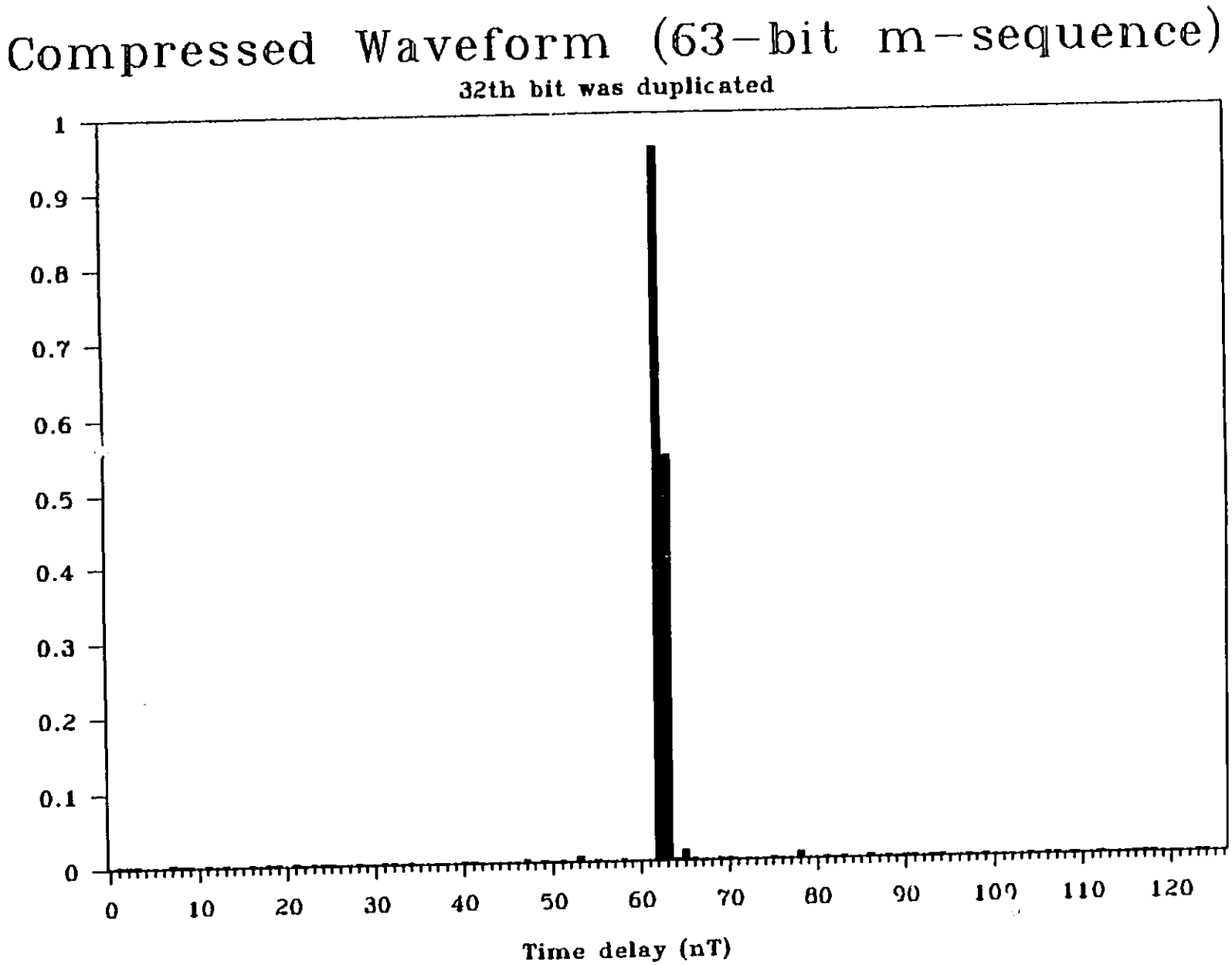


Fig. 4.13 Compressed waveform when the 32th bit of 63-bit m-sequence is duplicated.

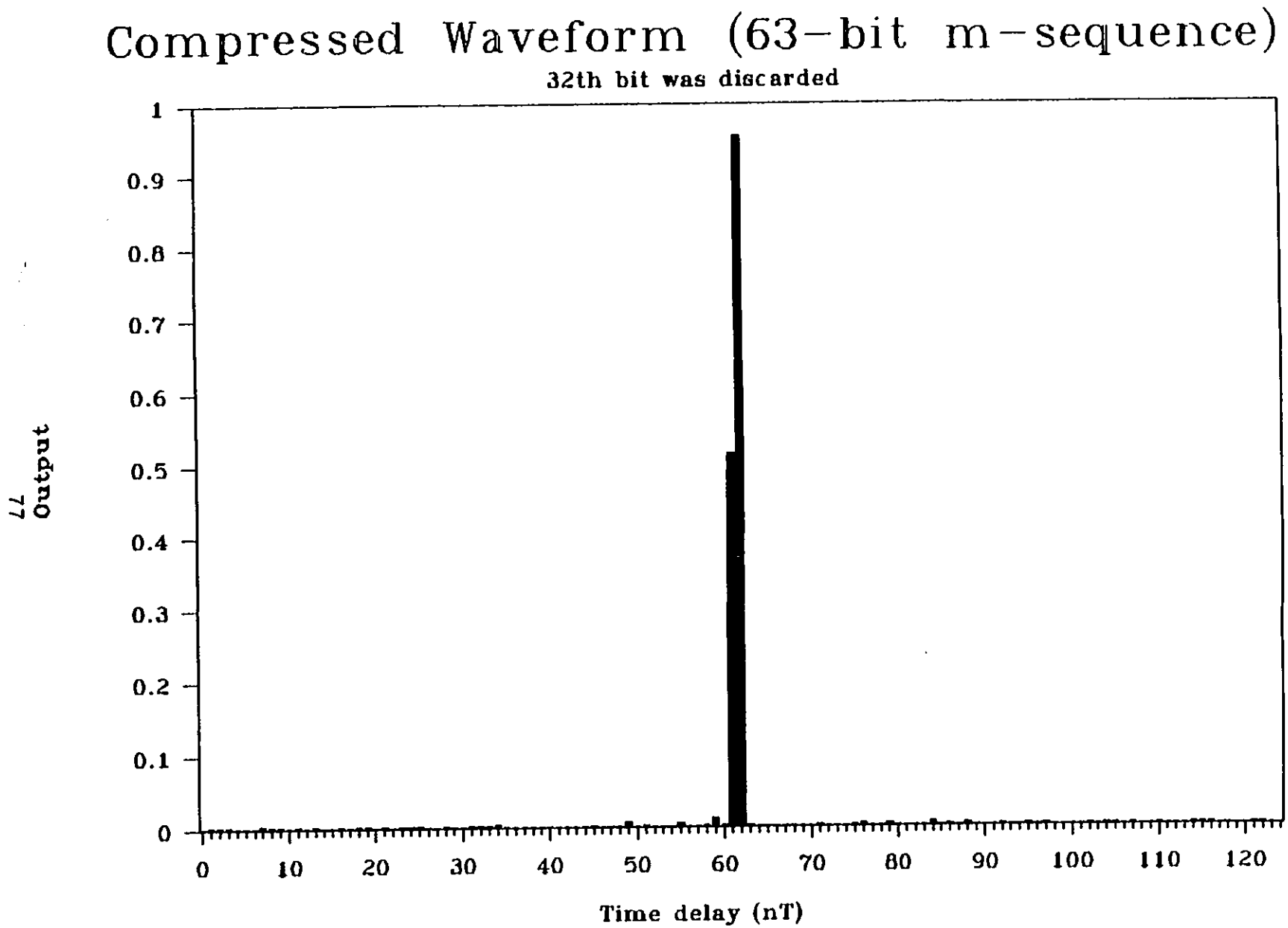


Fig. 4.14 Compressed waveform when the 32th bit of 63-bit m-sequence is discarded.

spike as the sidelobe value. As shown in this Table, the magnitude ratio decreases with the misaligned position further from the middle position. This is because the misalignment is treated as the overlapping of two waves in the network. If it occurs near the middle, the two waves have the same contribution to the output; otherwise, one of them will be dominant. Another observation is that the signal-to-sidelobe ratio does not form a trend when the position is different.

Table 4.3
Effects of the Position of Misaligned Bit Using 63-bit M-sequence

a. 1 bit is duplicated

	Bit to be duplicated						
	17th	22th	27th	32th	37th	42th	47th
Mag.Rat.	4.59	1.47	1.12	1.75	3.20	4.79	15.5
SNR(dB)	39.5	38.8	26.4	37.5	42.8	39.9	28.1

b. 1 bit is discarded

	Bit to be discarded						
	17th	22th	27th	32th	37th	42th	47th
Mag.Rat.	13.3	1.73	1.14	1.85	2.69	5.54	17.1
SNR(dB)	39.5	37.8	21.7	38.4	34.9	40.2	35.4

4.1.4 Effects of Received Signal Magnitude, Code length, Number of Hidden Neurons and Training Epochs

A. Effect of Received Signal Magnitude

The maximum magnitude of the compressed signal and the corresponding output signal-to-sidelobe ratio are plotted against the magnitude of the received input signal as shown in Fig. 4.15 and Fig. 4.16, respectively. From Fig. 4.15, one can notice that the network is non-linear as the non-linear activation function was used. Since for the back-propagation, the activation function is non-linear. From Fig. 4.16, we can find that the signal-to-sidelobe ratio depends on the magnitude of the received input signal. There is an optimum point when the magnitude is about ±0.6.

B. Effect of Code Length

Different code lengths (15, 31, 63) for the m-sequences have been used to find the effect of the code length on the signal-to-sidelobe ratio. The results are shown in Fig. 4.17. From Fig. 4.17, we find that the output signal-to-sidelobe ratio increases with the code length. It is due to the increase in the number of neurons in the input layer and hence the number of interconnections between any two adjacent layers of neurons. Thus the storage capacity of the network will be increased and hence a more parallel and distributed set of weights can be obtained which would give

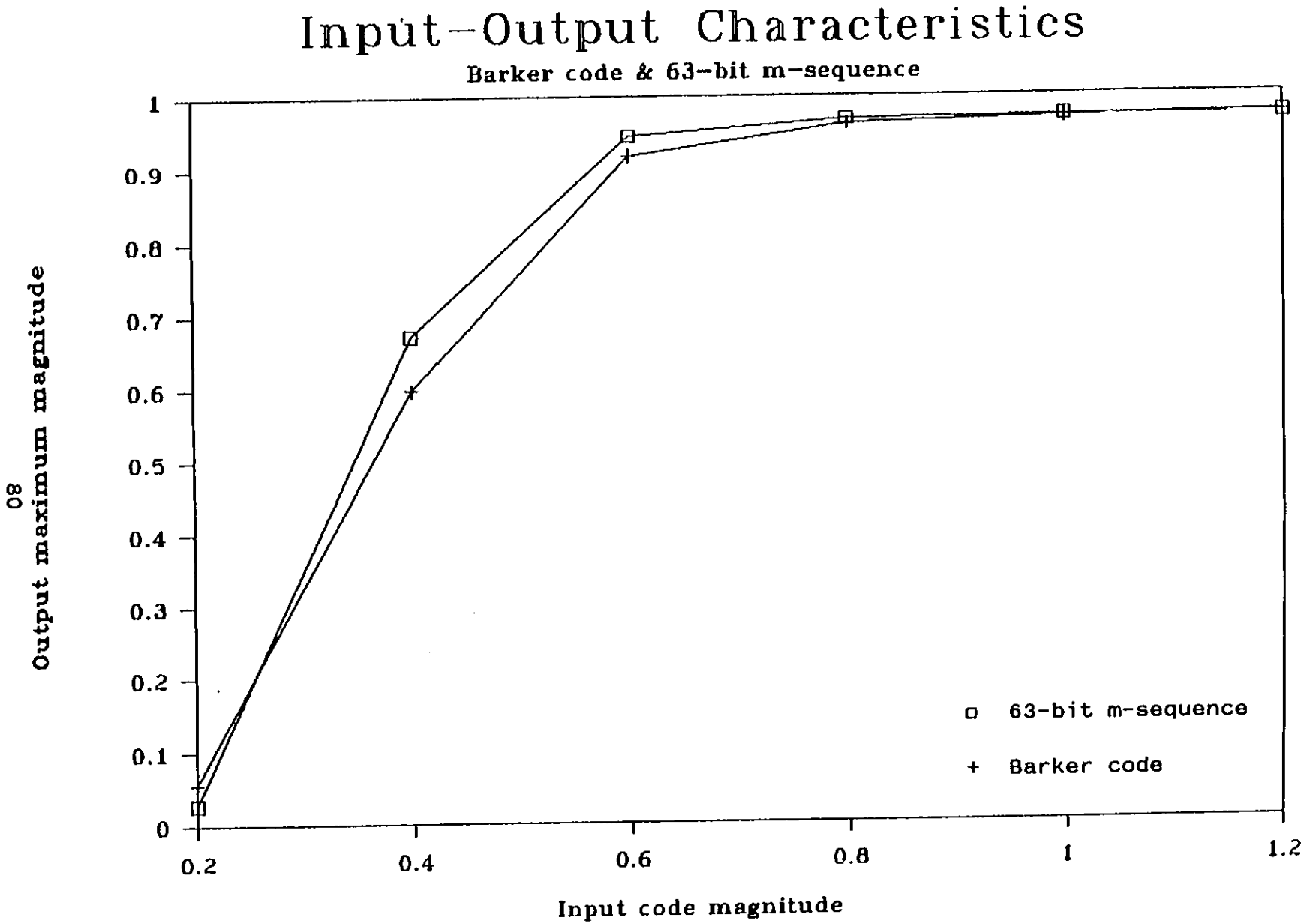


Fig. 4.15 Input-output characteristics of networks using Barker code and 63-bit m-sequence.

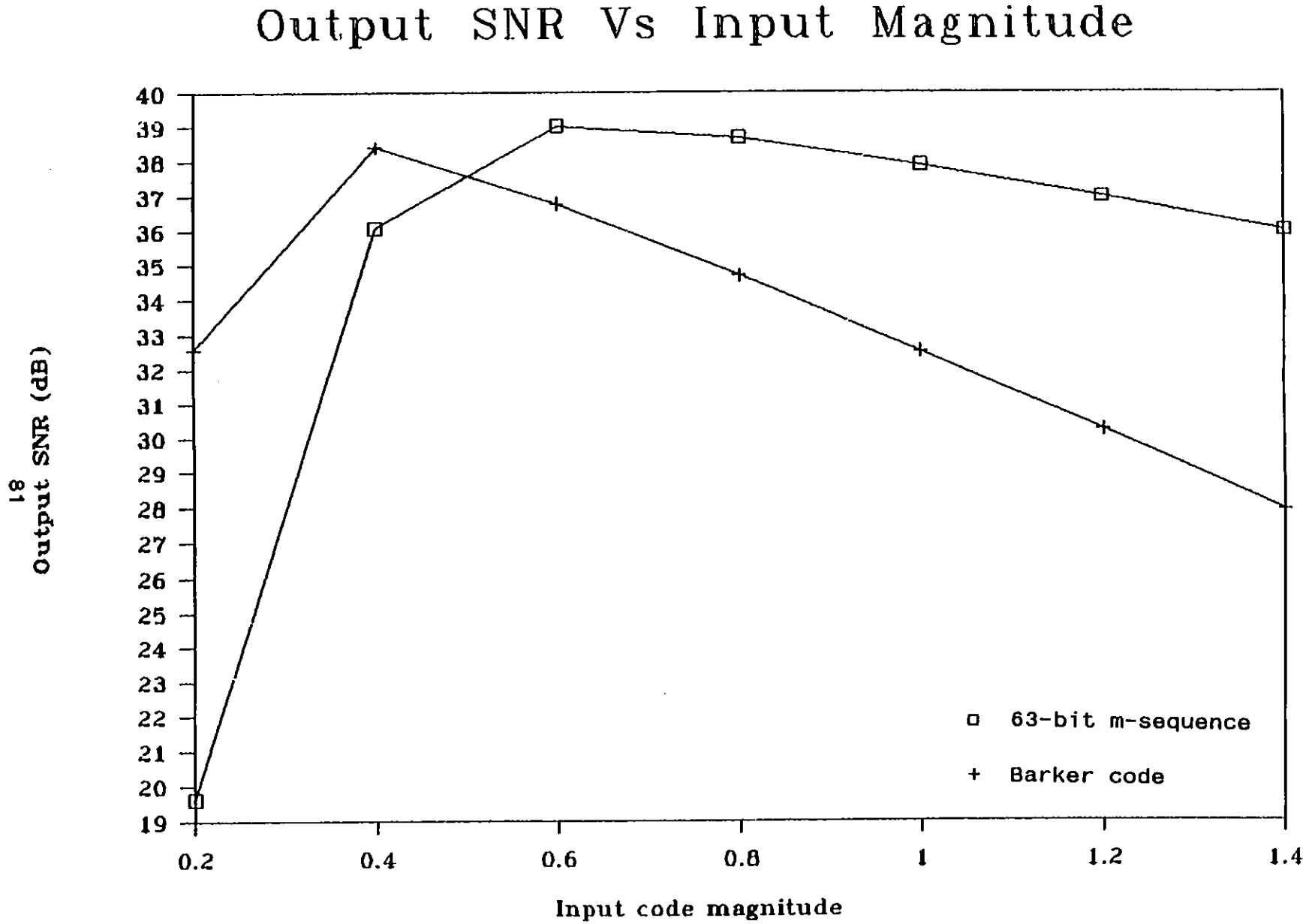


Fig. 4.16 Output signal-to-sidelobe ratio with different input code magnitude using Barker code and 63-bit m-sequence.

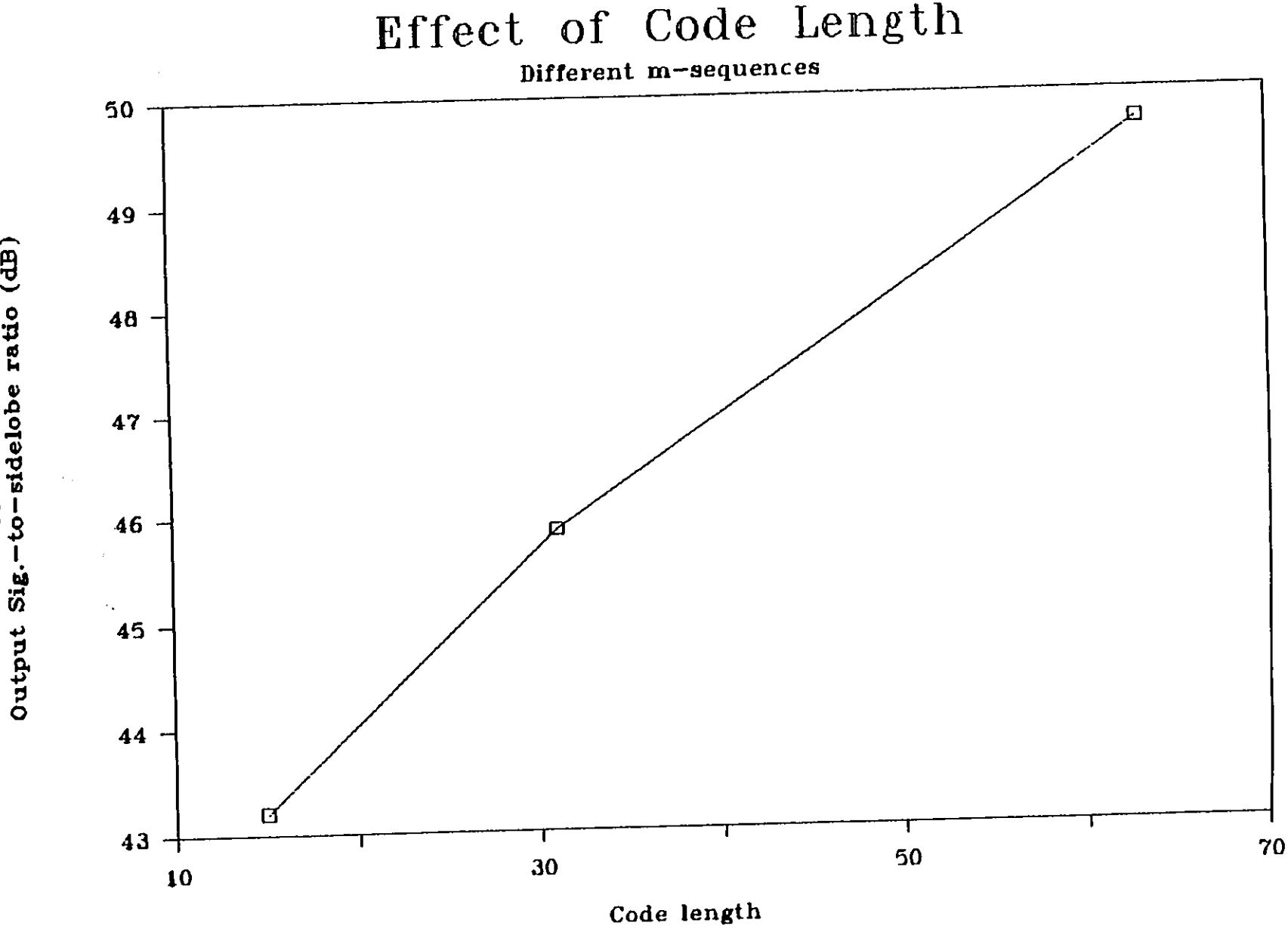


Fig. 4.17 Output signal-to-sidelobe ratio with different code length using m-sequences.

rise to a higher output signal-to-sidelobe ratio and consequently a more accurate detection. (This is similar to the fact that the higher the order of a filter, the better the shape of its magnitude response.) However, the longer the code length, the more the number of interconnections it has in the network. This makes the hardware implementation difficult. So, practically speaking, there is a limit to the length of the code that could be used for detection.

C. Effect of the Number of Hidden Neurons

Different numbers of hidden neurons have been used to find their effects on network performance. The results are shown in Table 4.4. The code used was the Barker code and the number of training epochs was 200. Although it was found that using more hidden neurons could have larger signal-to-sidelobe ratio, using three hidden neurons is a good choice. It is because more hidden neurons will result in a larger number of interconnections and will make the network more complicated. Secondly, if one or two hidden neurons were used, the system would not be robust enough. The robustness of this network has been tested by disabling one hidden neuron out of the three hidden neurons that were used. The result was that the network can still detect the received signal but the width of the compressed waveform is larger (similar to the output of misalignment simulation) which could degrade the accuracy of detection. Thus, the

present network does provide a minimum number of hidden neurons while providing quite a robust means for detection.

TABLE 4.4
Effects of the Number of Hidden Neurons

Number of Hidden Neurons	Signal-to-sidelobe (dB)
1	36.47
2	37.84
3	37.89
4	38.30
5	39.25

D. Effect of Training Epochs

For both the Barker code and the 63-bit m-sequence, the noise performances have been investigated when the number of training epochs was 200 (when the maximum output rms error is smaller than 0.05). The results are shown in Fig. 4.18 and Fig. 4.19, respectively. It was found that an increase in the number of training epochs will increase the signal-to-sidelobe ratio very much when the environment SNR is above 6dB and it will give a smaller increase in the signal-to-sidelobe ratio or even decrease the ratio when the environment SNR is well below 6dB.

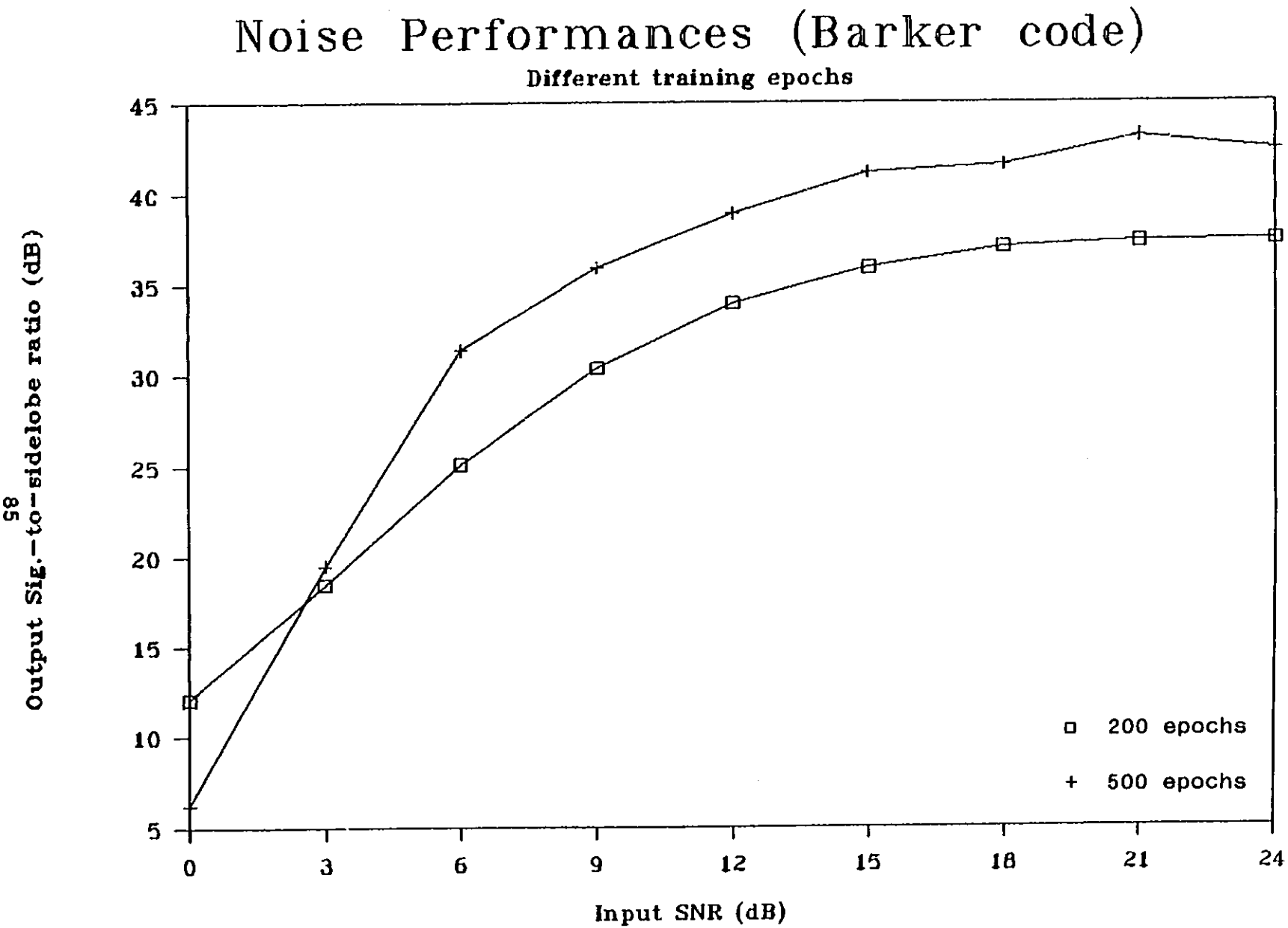


Fig. 4.18 Noise performances with different number of training epochs using Barker code.

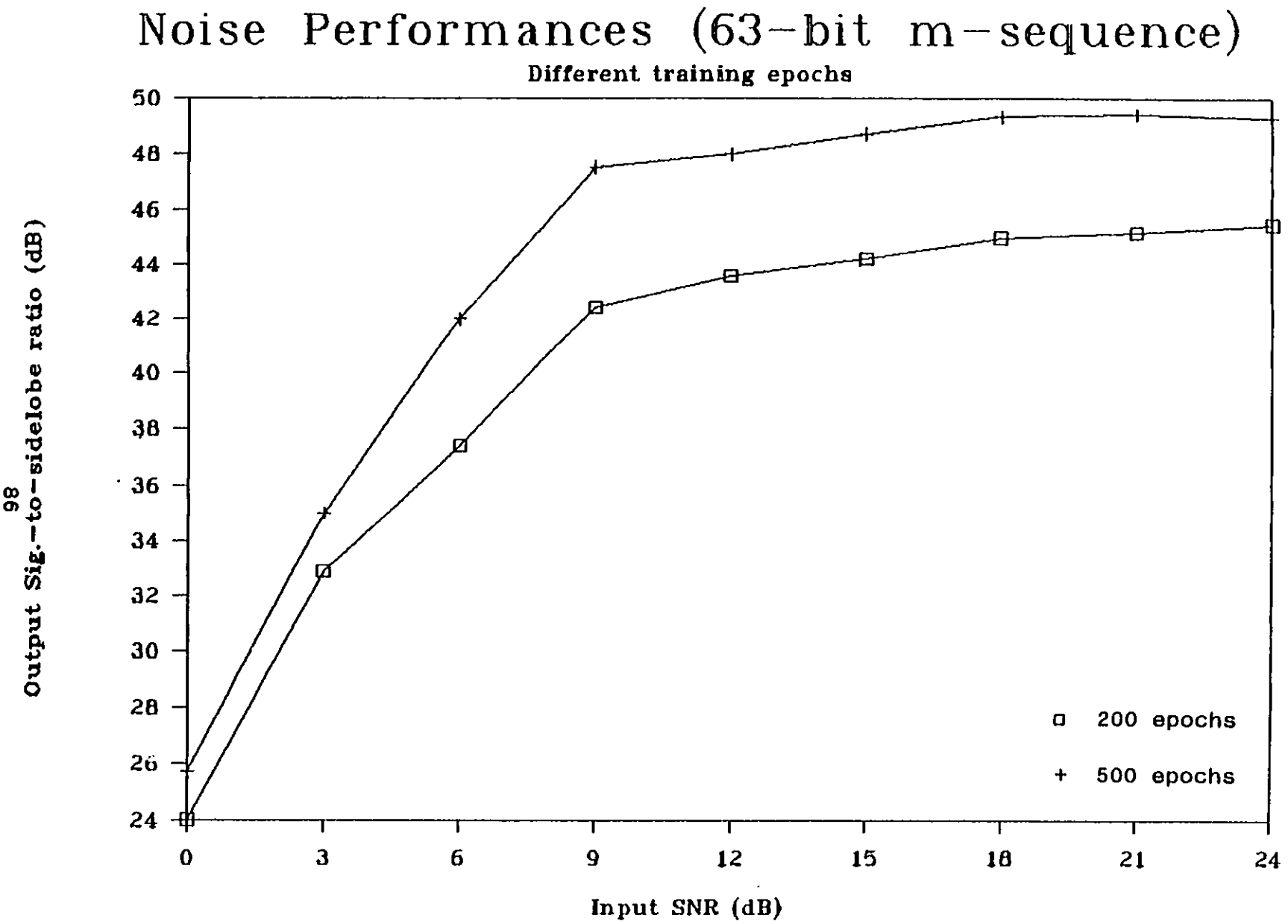


Fig. 4.19 Noise performances with different number of training epochs using m-sequence.

4.1.5 Summary

A neural network approach to the pulse compression has been studied and extensive simulations have been carried out to examine the performance of the approach. This new approach has a much superior output signal-to-sidelobe ratio than those of the traditional approaches such as the correlator and the inverse filters. Moreover, since a large number of connections have been used, the network is much more robust than the traditional approaches. The robustness was tested by disabling one neuron (please refer to 4.1.4 C). However, this approach has a drawback of non-linearity which can be overcomed by putting a compensation device at the output. To sum up, it is believed that the proposed method is of practical significance to those who are involved with the implementation of pulse radar detection system.

4.2 USE IN FILTERING

Another interesting application is to use this non-linear network to do filtering. There are many traditional approaches, such as the use of optimization and the use of some well known filters such as elliptic filters, to design a filter. These approaches are already good enough for designing a filter but it may still be useful to study the neural network approach which may provide a method of least human design effort -- there is no need to write a program and just let the neural network do the learning. Moreover, the neural network approach is more flexible in that one can train the network with desire frequency spectrum of any shape.

4.2.1 The Neural Network Approach

As in pulse compression, a 3-layer feed-forward network using the back-propagation rule is used. The activation function is the sigmoid function and there is only one output unit which is scaled to the range (-2, 2) for filtering the sine waves which has a magnitude ±1. Since a sigmoid function cannot have an output in the negative range, the sigmoid output should be multiplied by 4 and then is subtracted by 2. Different combinations of the number of input units and the number of hidden units are used and the results are compared. The training is done by shifting dif-

ferent sampled analog frequencies to the input unit so that all parts of a sine wave can be learned. The sine waves are generated by a program WAVEGEN.C which can output sine waves with or without distortion, with or without phase shifted and two added sine waves. As in pulse compression, the output is in Network Professional II format. The objective of this research is to have a lowpass filter with a cut-off frequency at 4kHz. The sampling frequency used is 16kHz. Altogether, 16 different frequencies, from 0 to 7.5kHz with 500Hz interval, are presented for training. For different frequencies, the number of training data sets is different since the number of sampling points required for a complete sampling of the corresponding frequency is different. The number of training data sets is as follows: 0Hz - 1 training data, 4kHz - 4 training data, 2kHz and 6kHz - 8 training data, 1kHz, 3kHz, 5kHz and 7kHz - 16 training data, 500Hz, 1.5kHz, 2.5kHz, 3.5kHz, 4.5kHz, 5.5kHz, 6.5kHz and 7.5kHz - 32 training data. The target output is equal to the value of the last input neuron if the input frequency is below or equal to 4kHz and is zero if the input frequency is higher than 4kHz. After 200 epochs of training, the network performs well for the trained frequencies. Two examples of input-output waves are shown in Fig. 4.20 and Fig. 4.21. In the following section, the properties of this network using 16 input neurons, 32 hidden neurons and 1 output neuron are discussed. It was found that this network had a better

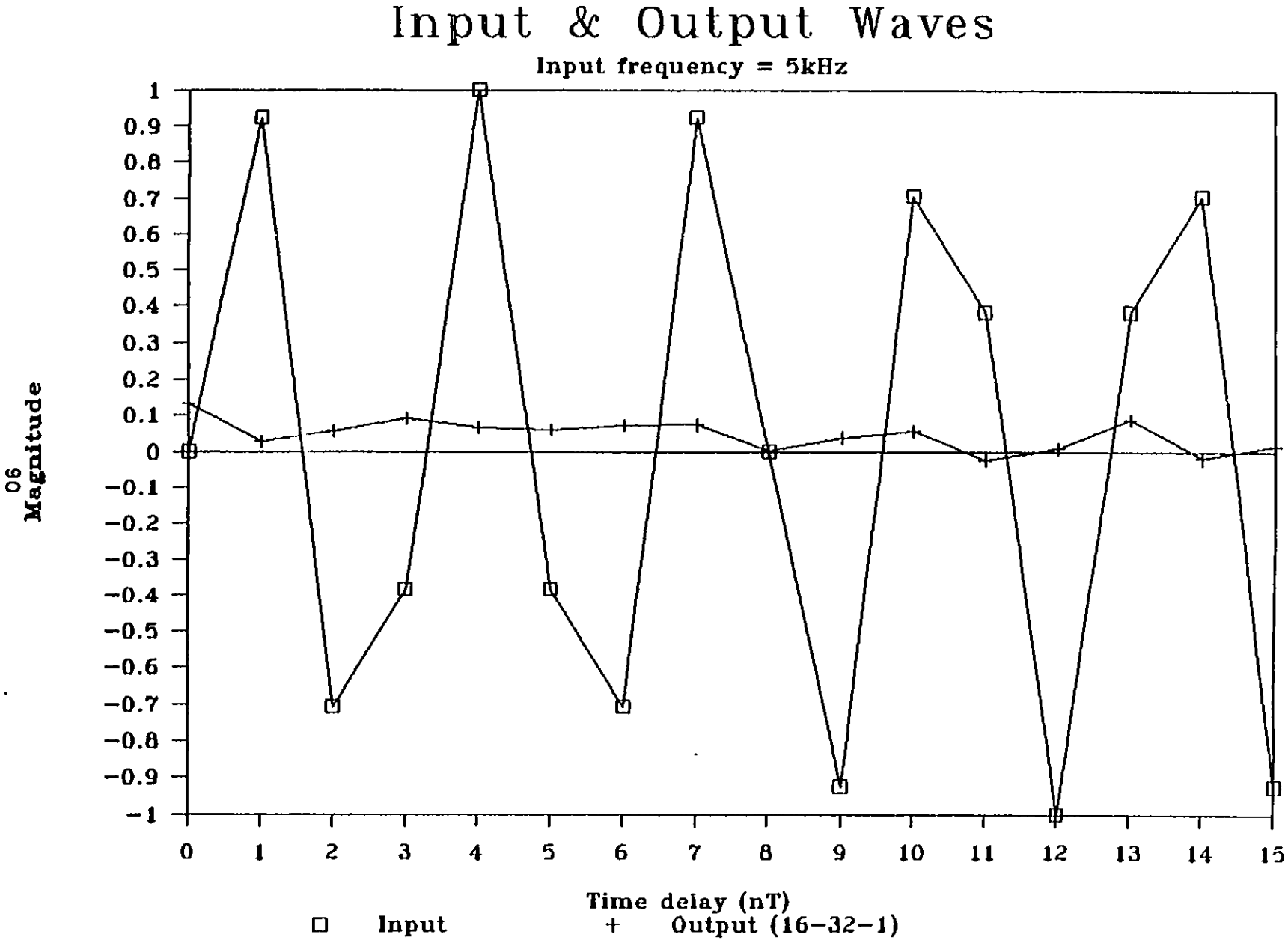


Fig. 4.21 Output of the network together with the 5kHz input.

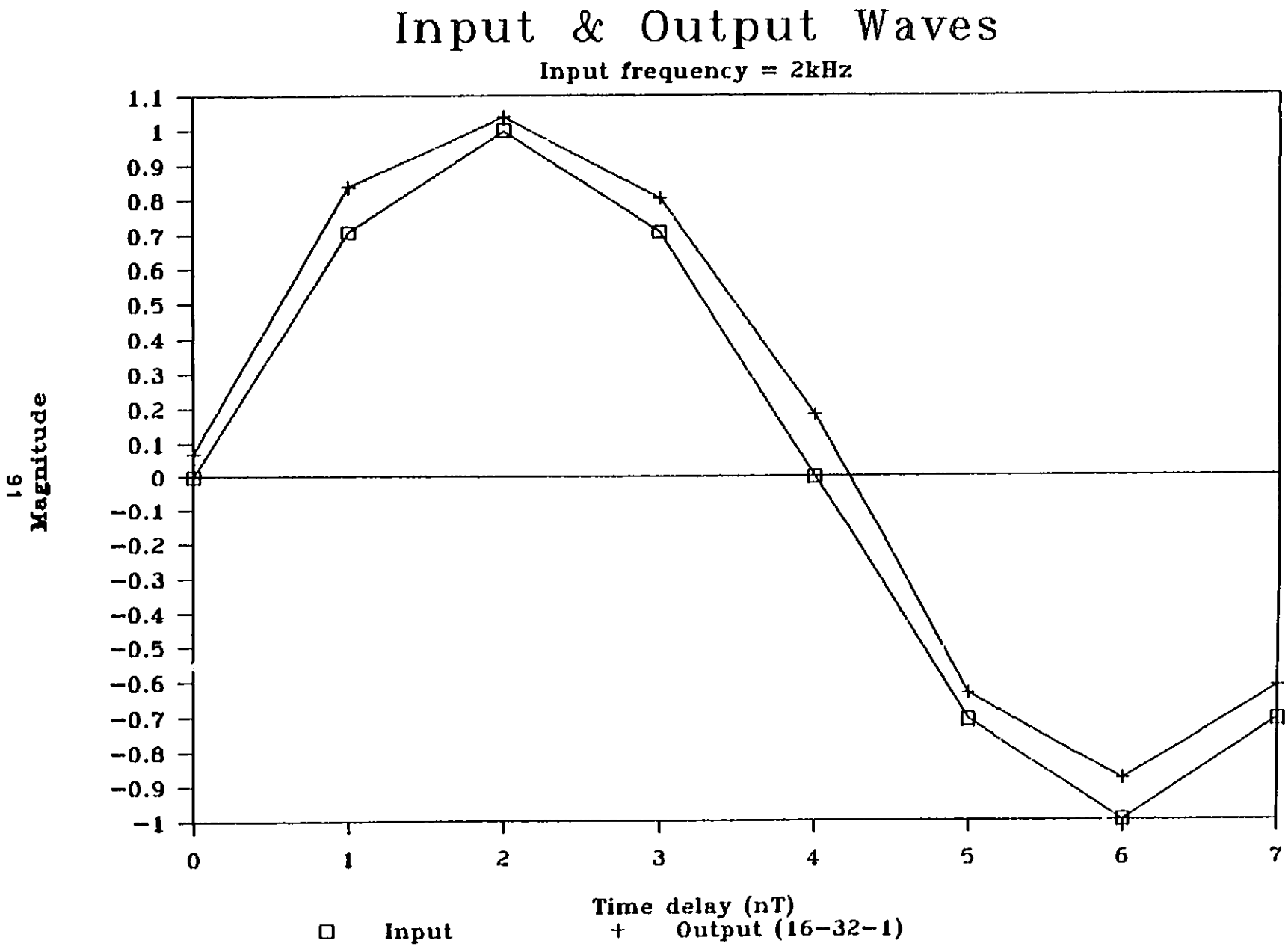


Fig. 4.20 Output of the network together with the 2kHz input.

recognition of untrained frequencies and thus it is suitable as a filter.

4.2.2 Properties

A. Recognition of Phase Shifted Frequency

Since, in presenting the input frequency, the sampling points are just part of the signal especially in high frequency, a phase shifted version of a 2kHz sine wave was presented to the network to test whether the network could recognize the signal. This phase shifted version does not consist of the points that are trained. As shown in Fig. 4.22, the network successfully recognized this frequency.

B. Output for Addition of Two Waves

Two sine waves were added and presented to the network to test its additive ability. The following combinations were tried: 500Hz + 1kHz (both in passband), 2kHz + 5kHz (one in passband and one in stopband) and 6kHz + 7kHz (both in stopband). The results are shown in Fig. 4.23 to Fig. 4.25. They show that the network has additive ability. It can extract a low frequency from a high frequency.

C. Output for Untrained Frequencies

Some untrained frequencies were presented to the network to test whether the network could behave like a lowpass filter for all frequencies. Two examples, 200Hz and 6.8kHz

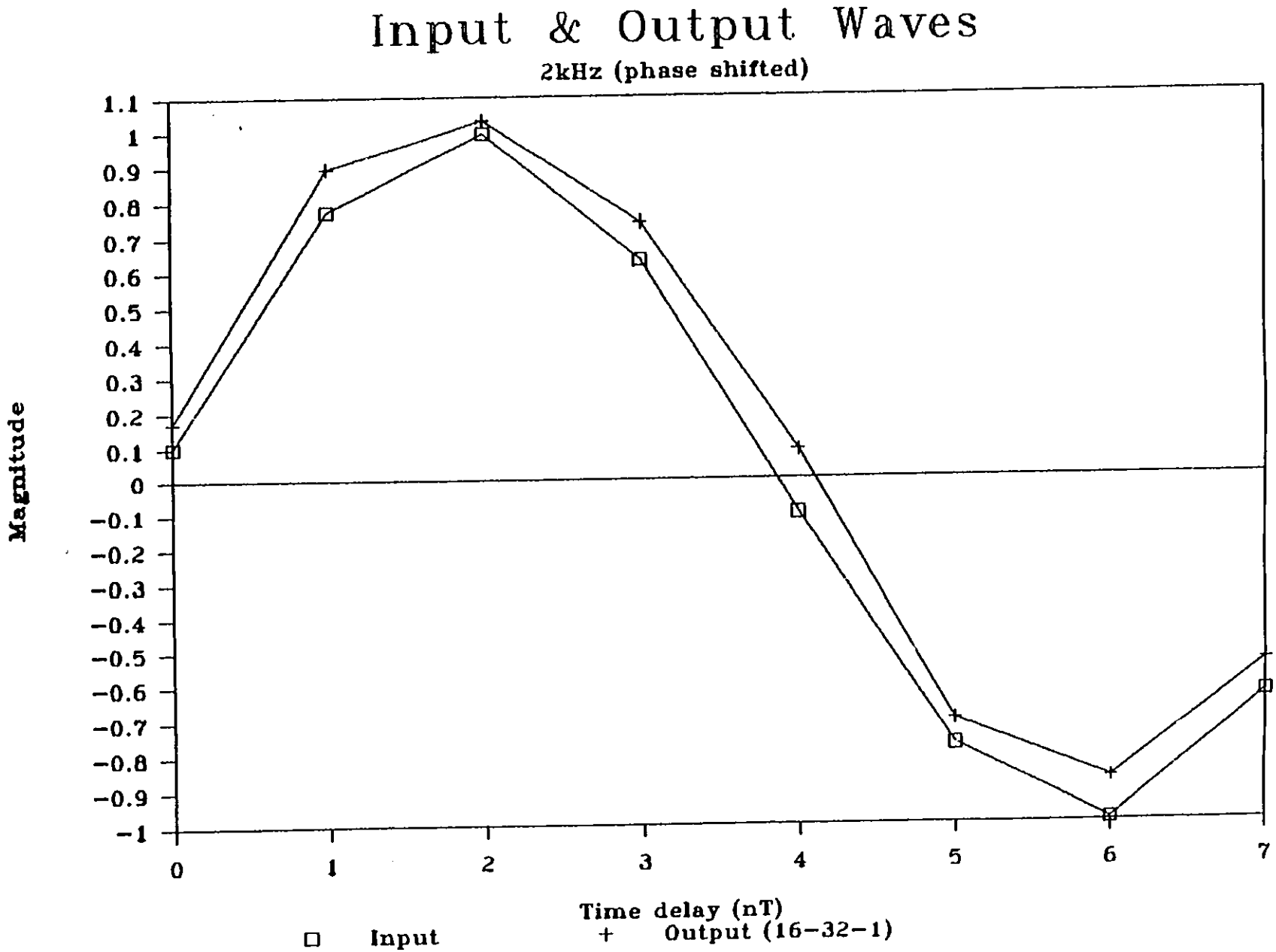


Fig. 4.22 Output of the network together with the 2kHz input(phase-shifted)

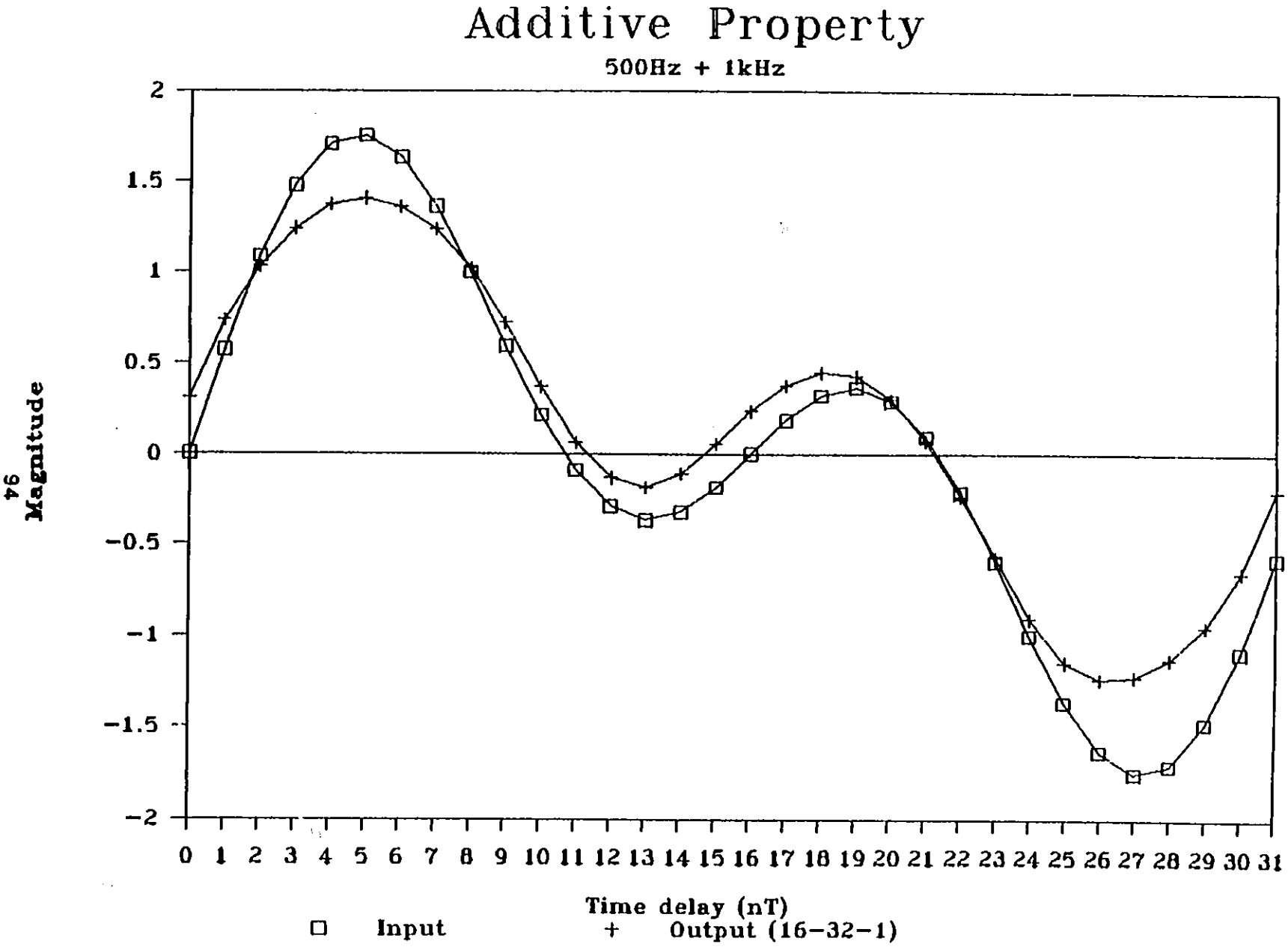


Fig. 4.23 Output of the network together with the 500+1kHz input.

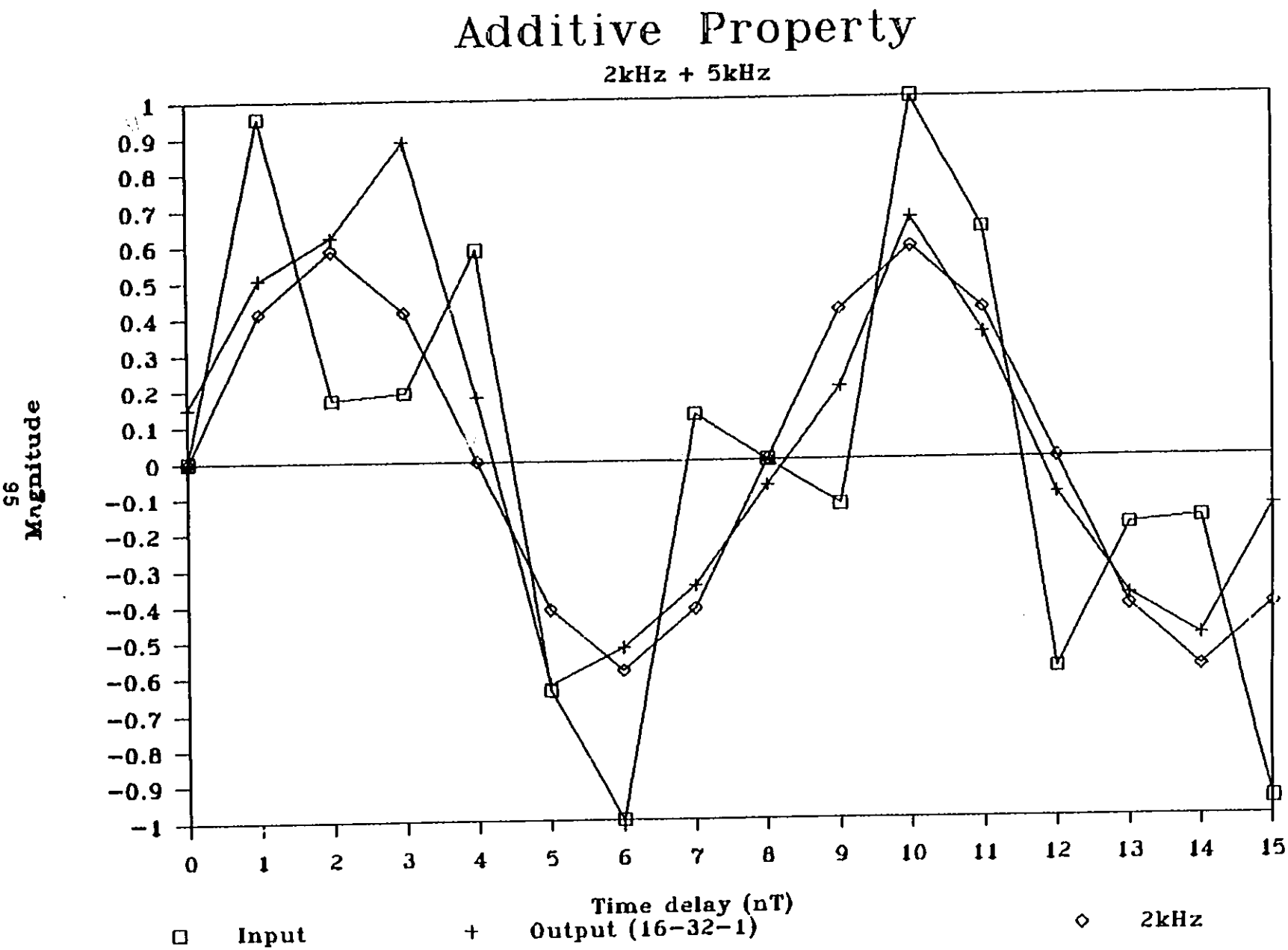


Fig. 4.24 Output of the network together with the 2k+5kHz input.

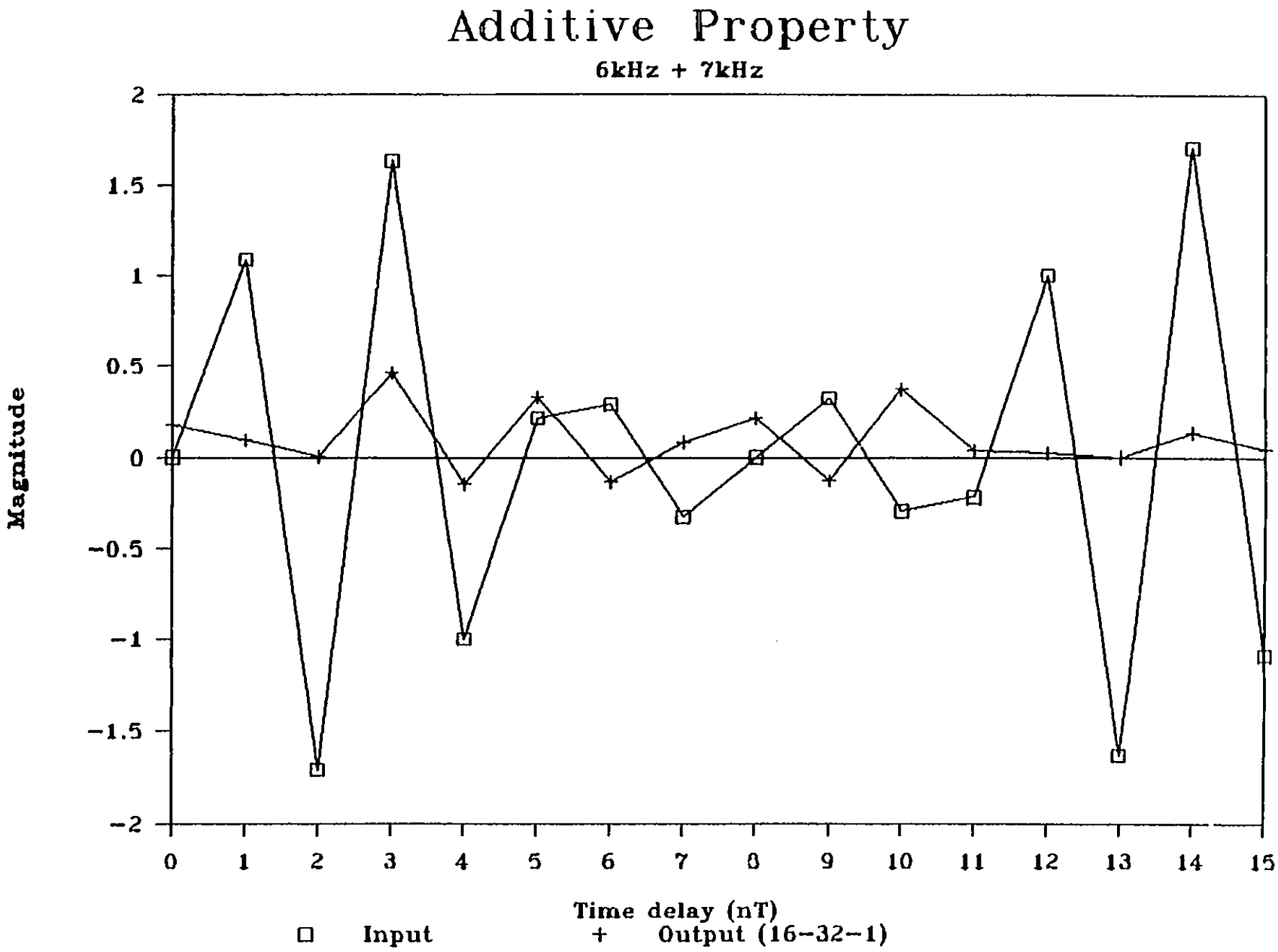


Fig. 4.25 Output of the network together with the 6k+7kHz input.

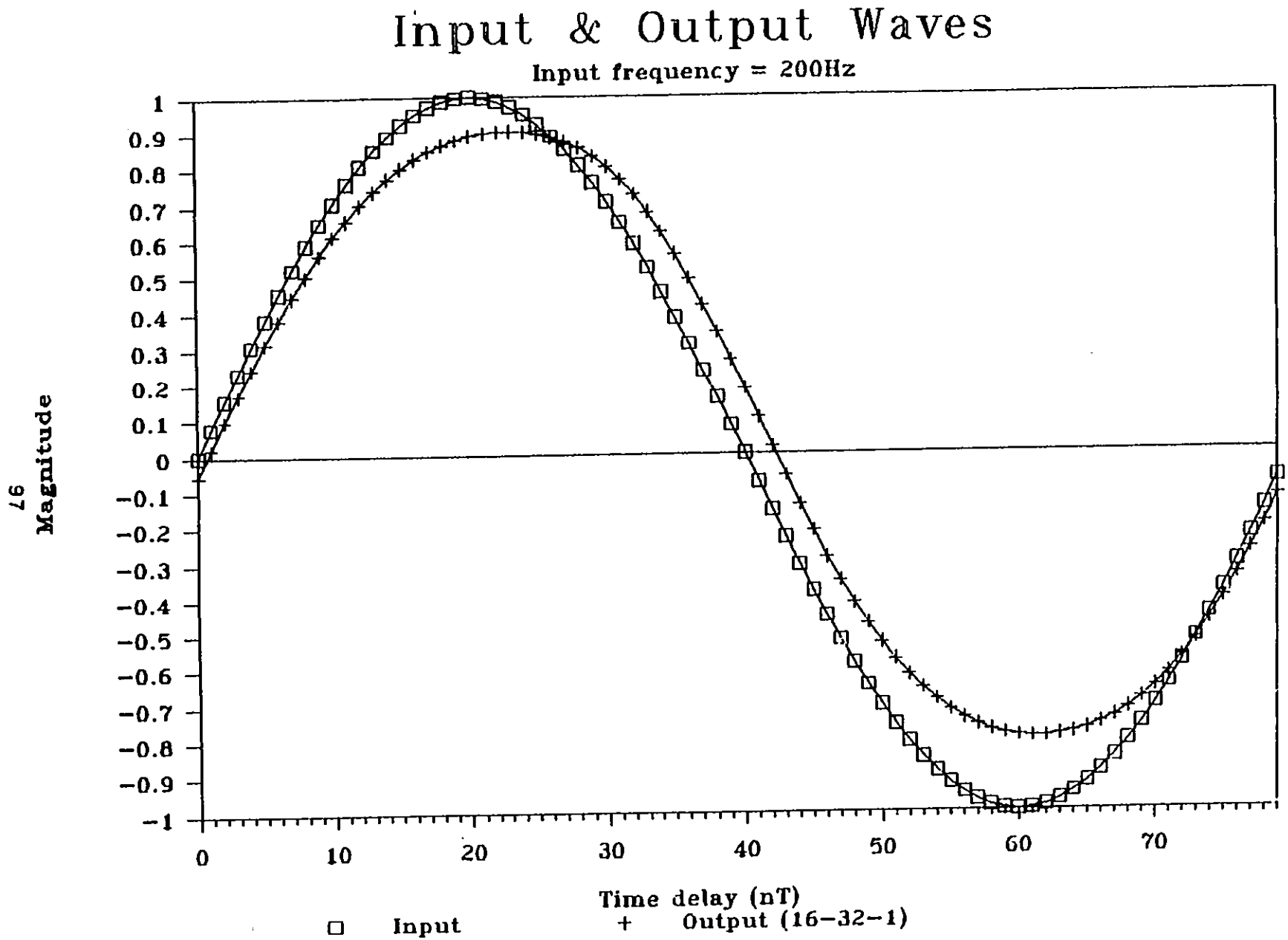


Fig. 4.26 Output of the network together with the 200Hz input.

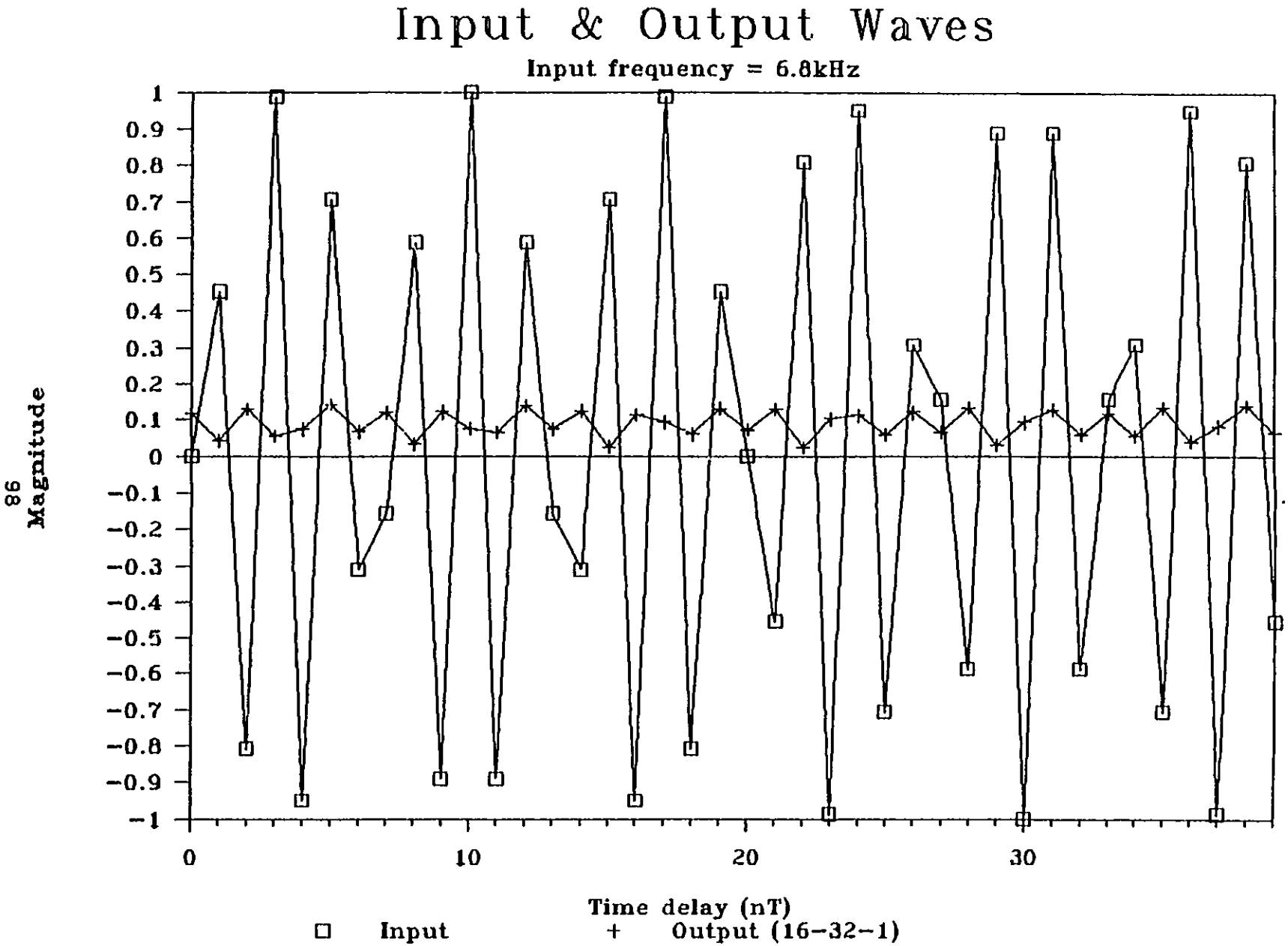


Fig. 4.27 Output of the network together with the 6.8kHz input.

are shown in Fig. 4.26 and Fig. 4.27. As shown in these graphs, the network can generalize the output to the frequencies that were not used to train the network.

D. Effect of the Number of Input Neurons

Fixing the number of hidden neurons to 16 (which is the number of sine waves in training), the number of input neurons was changed from 8 to 32 to find out the effect of this number. The total root mean square errors of each trained frequencies were found and plotted in Fig. 4.28 with the number of epochs fixed to 200. Three typical untrained frequencies (200Hz for passband, 4.3kHz for transition band and 6.8kHz for stopband) are also plotted in the same graph. Results show that the higher the number of input neurons, the higher the accuracy the output of the trained frequencies are. However, increasing the number of input neurons cannot increase the generalization of the network, i.e., the network cannot recognize the untrained frequencies even though the number of input neurons is increased. An example of untrained 200Hz testing using a 32 input neurons, 16 hidden neurons and 1 output neuron network is shown in Fig. 4.29.

E. Effect of the Number of Hidden Neurons

Fixing the number of input neuron to 16 (the best for untrained frequencies), the number of hidden neurons

was changed from 16 to 64. Again the total root mean square errors were found and plotted in Fig. 4.30 for 200 epochs training. Results show that increasing the number of hidden neurons does not guarantee that the accuracy of the output will be higher for both the trained and the untrained frequencies.

4.2.3 Summary

Some of the results of using neural network to do low-pass filtering was discussed. It is interesting to find out that the network can recognize the untrained frequencies. The results on the effects of the number of input neurons and the number of hidden neurons are also useful for further study in this subject.

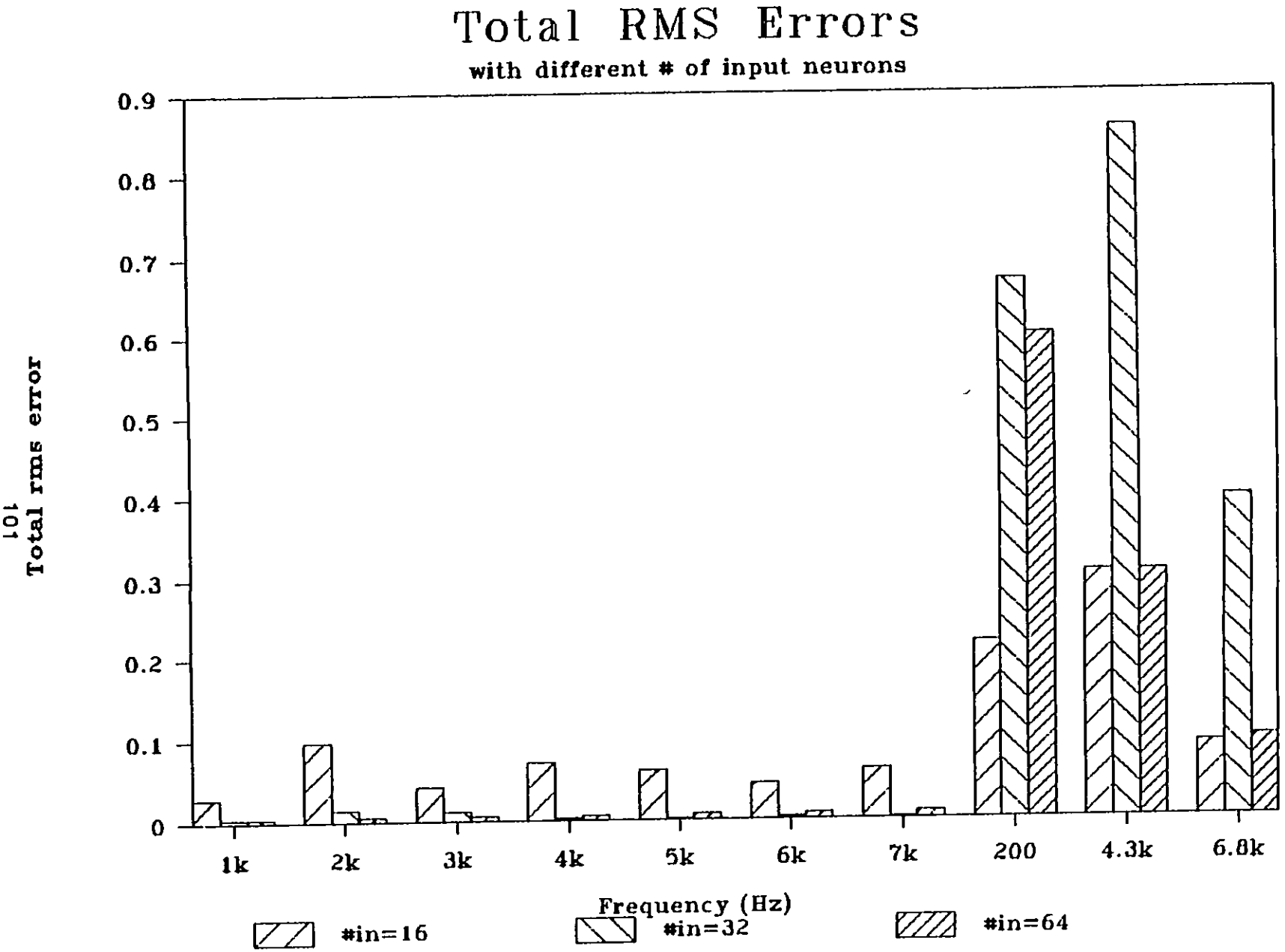


Fig. 4.28 The effect of the number of input neurons.

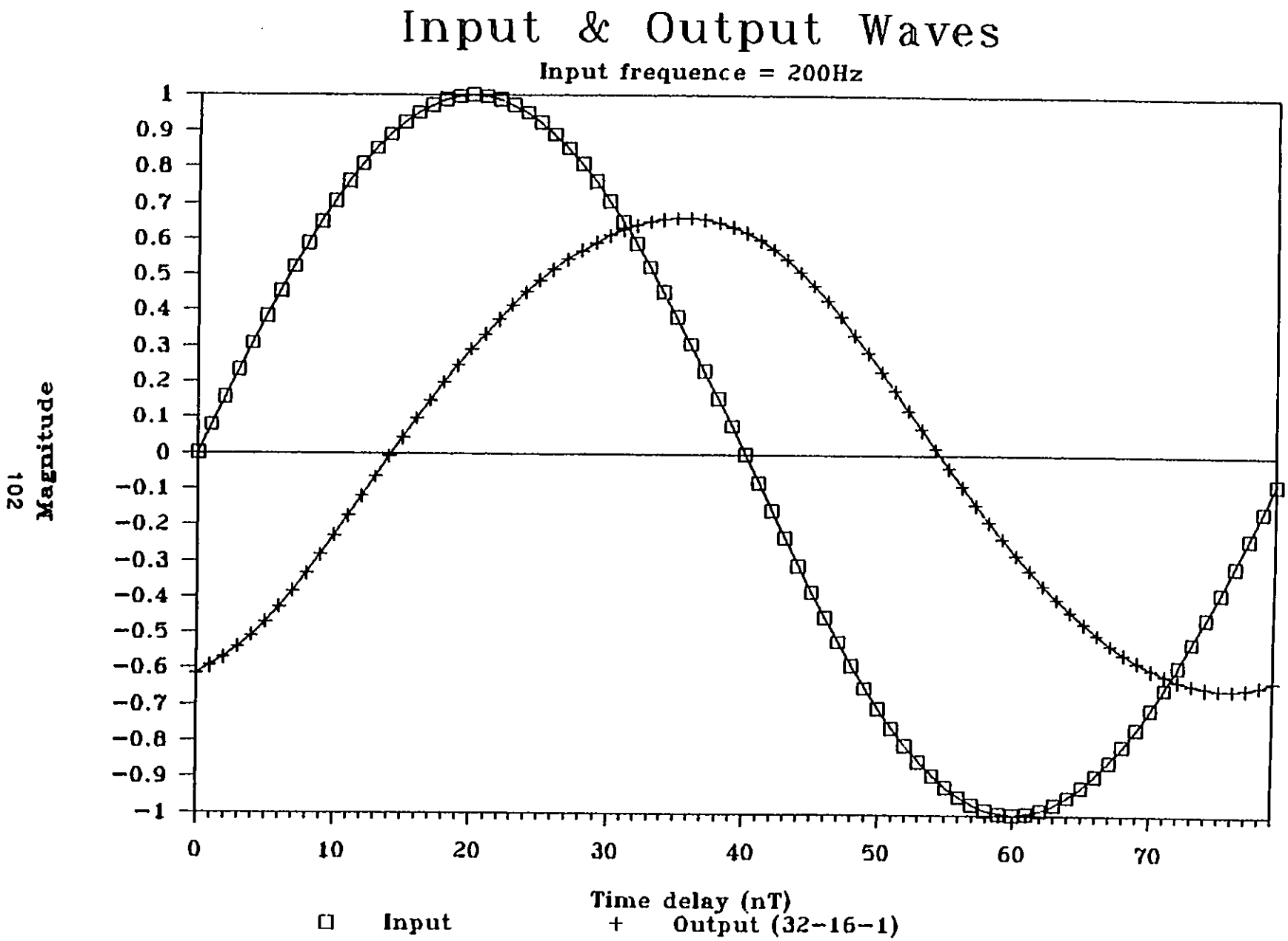


Fig. 4.29 Output of (32-16-1) network together with the 200Hz input.

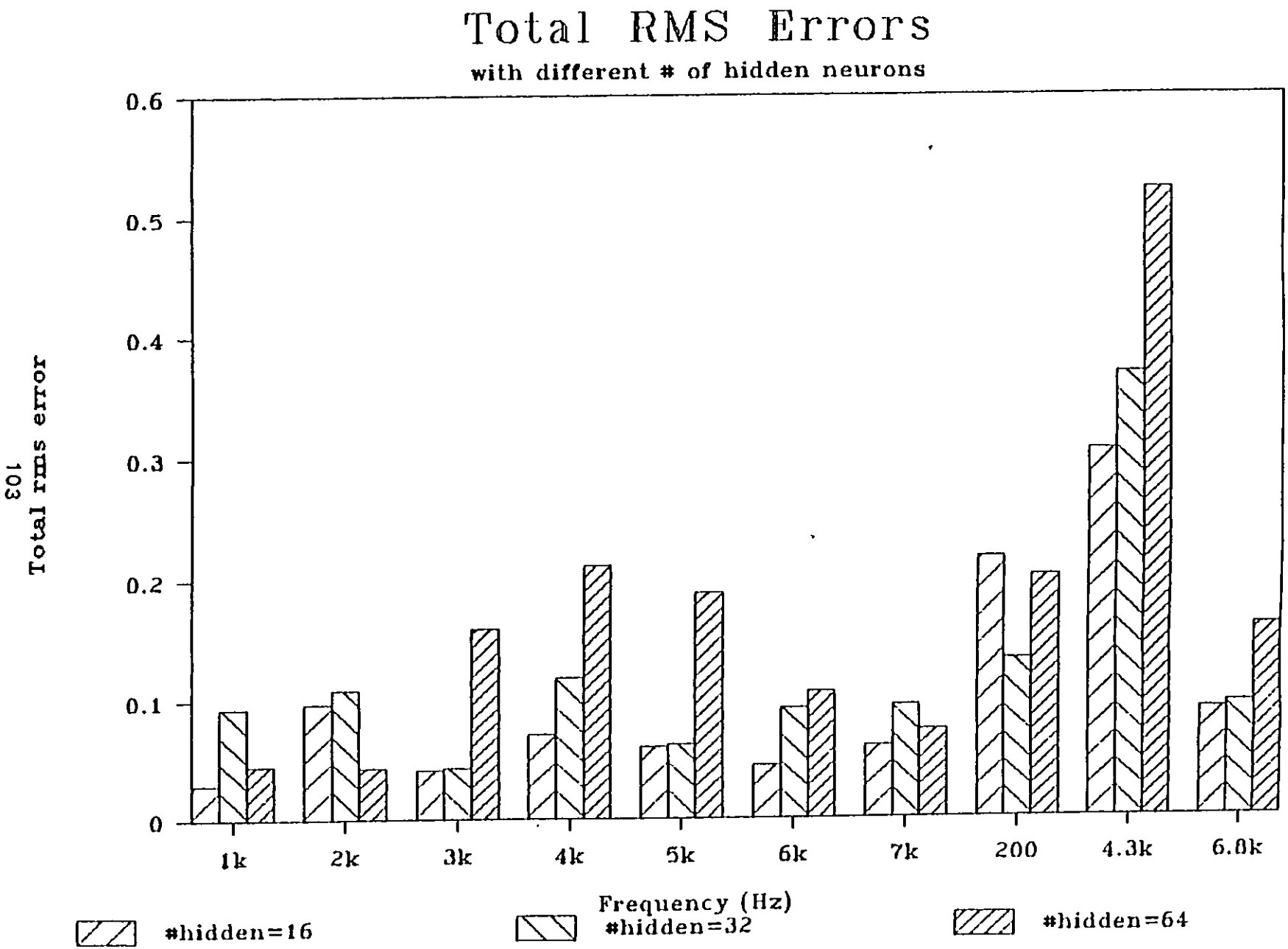


Fig. 4.30 The effect of the number of hidden neurons.

CHAPTER V CONCLUSIONS

In this thesis, two important issues in neural network are discussed. The first one, which is discussed in Chapter III, is the storage capacity of neural networks. The design of a simple learning rule, which can provide a higher storage capacity, to train the neural networks is the first objective of this thesis. In Chapter III, the problem of storing temporal associative patterns is addressed. Two new network configurations, the ring and the cascade networks, are introduced. They successfully store and recall temporal patterns.

Two new methods for increasing the storage capacity of a single layer network are also introduced in Chapter III. The first one is the modified Hebb rule followed by two algorithms for different vector lengths. This method is found not to be useful for hetero-associative data (please refer to 3.3.6). In fact, the modified Hebb rule has a drawback in that it will greatly increase the value of weights when the L.C.M. is very large, especially when many patterns are stored. This drawback can be solved by normalizing the weights. Since this method can still greatly increase the

storage capacity of a single layer network, it is useful for storing auto-associative data. The second method is introduced to tackle the problem of storing hetero-associative data. It uses the idea of multi-threshold values which increase the number of hyperplanes in one neuron. This method is found to be very effective in storing the data (please refer to 3.3.6). However, one problem arises: will it be stable when multi-threshold values are introduced? Although a formal analysis of the stability of this kind of network is not given, its stability is supported by the simulation results obtained. This method appears to be an important result in the storage capacity of neural network. Another aspect to be pointed out here is that the noise performance was found to be degraded when more data were stored. So there is a compromise between the number of data stored and the noise performance of the network. If too much data are stored, the error correction ability of the neural network will be deterioated.

The second objective of this thesis is to apply the neural network approach to the signal processing problem. The first one is the pulse compression which can be done by correlator and inverse filter. Detailed results are found and presented in Chapter IV. In this research, it is interesting to find that the neural network has an additive property. This property is useful in many signal processing

problems. Moreover, the noise performance of the neural network approach is found to be much better than the traditional approaches. So the neural network approach has been successfully applied to pulse compression.

In Chapter IV, another signal processing problem is addressed. It is the lowpass filtering. The properties of this lowpass filter were found. Again, the neural network approach is found to have the additive property. Furthermore, the network is found to have a generalization property. It can recognize the frequency that is not trained. Although the use of the neural network is worse than the traditional approach in the shape of output waveform, the properties found in this part of the thesis are also useful for further studies.

In conclusion, some recommendations for further researches are given here:

1. A formal analysis of the storage capacity of using the new algorithms should be done as this is quite important when using an algorithm.
2. Comparisons between the single layer and the multi-layer network can also be done to form a guideline of what network should be used for a particular problem.
3. Following the comparisions, feasible application of a single layer network to pulse compression and filtering

can be studied.

4. Other new neural network applications, such as those in the area of signal processing, can be tried. In fact, there are many other applications which can be done using neural networks.

REFERENCES

[1] Ackroyd, M. H. and Ghani, F "Optimum Mismatched Filters for Sidelobe Suppression", IEEE Transactions on Aerospace and Electronic Systems, AES-9, 2, March 1973, p. 214-218.

[2] Duda, R. O., and Hart, P. E., "Pattern Classification and Scene Analysis", John Wiley and Sons, New York, 1973.

[3] Grossberg, S., "The Adaptive Brain I: Cognition, Learning, Reinforcement, and Rhythem, and the Adaptive Brain II: Vision, Speech, Language, and Motor Control", Elsevier/North-Holland, Amsterdam, 1986.

[4] Hebb, D. O., "The Organization of Behavior", John Wiley and Sons, New York, 1949.

[5] Hopfield, J. J., "Neural Networks and Physical Systems with Emergent Collective Computational Abilities", Proceedings of the National Academy of Sciences, USA, Volume 79, April 1982, p. 2554-2558.

[6] Hopfield, J. J., "Neurons with Graded Response Have Collective Computational Properties Like Those of Two-state Neurons", Proceedings of the National Academy of Sciences, USA, Volume 81, May 1984, p. 3088-3092.

[7] Hopfield, J. J. and Tank, D. W., "Computing with Neural Circuits: A Model", Science, Volume 233, August 1986, p. 625-633.

[8] Kohonen, T., "Self-Organization and Associative Memory", Springer-Verlag, Berlin, 1984.

[9] Kosko, B., "Bidirectional Associative Memories", IEEE Transactions on System, Man, and Cybernetics, SMC-18, No. 1, January/February 1988, p. 49-60.

[10] Lippmann, R. P., "An Introduction to Computing with Neural Nets", IEEE Acoustic, Speech, and Signal Processing Magazine, Volume 2, p. 4-22, April 1987.

[11] McCulloch, W. S. and Pitts, W., "A Logical Calculus of the Ideas Imminent in Nervous Activity", Bulletin of Mathematical Biophysics, 5, 1943, p. 115-133.

[12] McEliece, R. J., Posner, E. C., Rodemich, E. R., and Venkatesh, S. S., "The Capacity of the Hopfield Associative Memory", IEEE Transactions on Information Theory, Volume IT-33, p. 461-482, 1987.

[13] Mese, E. D. and Giuli, D., "Optimal Recursive Phase-coded Waveform Radars", IEEE Transactions on Aerospace and Electronics Systems, AES-13, 2, March 1977, p. 163-171.

[14] Minsky, M. and Papert, S., "Perceptrons", MIT press, Cambridge, Massachusetts, 1969.

[15] Nathanson, F. E., "Radar Design Principles", McGraw-Hill, New York, 1969, p. 452-469.

[16] Rosenblatt, F., "Principles of Neurodynamics", Spartan Books, New York, 1962.

[17] Rumelhart, D. E., McClelland, J. L. and the PDP Research Group, "Parallel Distributed Processing (PDP): Explorations in the Microstructure of Cognition. Volume 1: Foundations", MIT Press, Cambridge, Massachusetts, 1986.

[18] Sietsma, J. and Dow, R. J. F., "Neural Net Pruning", Proceedings of IEEE International Conference on Neural Networks, San Diego, California, July 1988, Volume I, p. 325-333.

[19] Widrow, G., and Hoff, M. E., "Adaptive Switching Circuits", Institute of Radio Engineers, Western Electronics Show and Convention, Convention Record, Part 4, August 1960, p. 96-104.

[20] "NeuralWorks Including An Introduction to Neural Computing, NeuralWorks User's Guide, Networks I, Networks II", NeuralWare Incorporation, PA.

[21] Lee, C. K., "Programs for Neural Network Simulations", 1990.

VITA AUCTORIS

Chi Kin Lee was born in 1965 in Macau. He graduated from Queen's College, Hong Kong in 1985. From there he went on to the University of Hong Kong, Hong Kong, where he obtained a Bachelor of Science (Eng.) degree in Electrical and Electronics Engineering in 1988. He is currently working toward his Master of Applied Science degree in Electrical Engineering at the University of Windsor, Windsor, Canada and hopes to graduate in May 1990.